Introduction to Python programming

Machine Learning in Python: Supervised Learning part 2

- Armando Camerlingo

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani

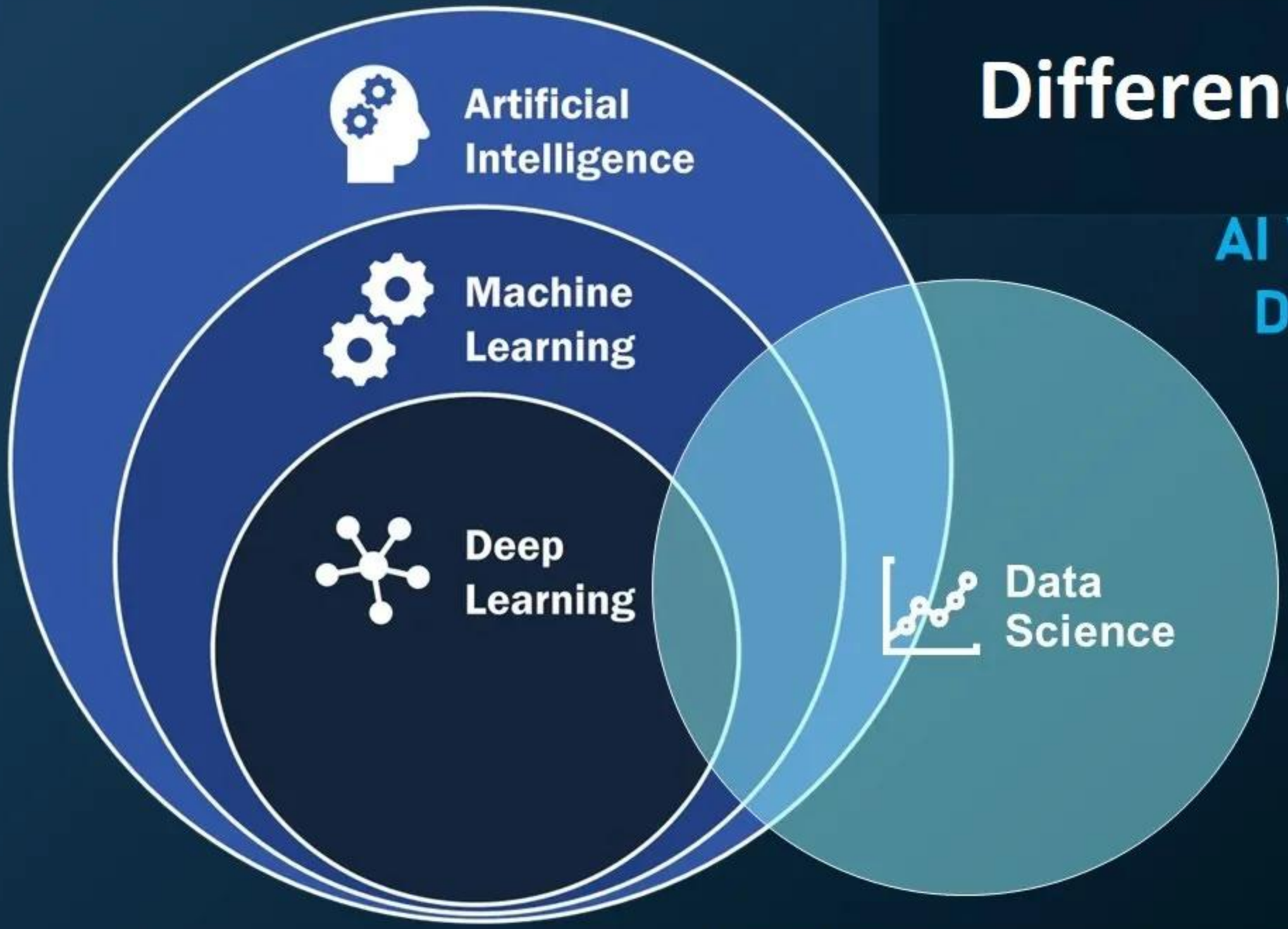
Introduzione alla Programmazione in Python

Armando Camerlingo

16/06/2025 - Lezione 4

Difference Between

AI VS ML VS DL VS DS



https://medium.com/@shahinaperveen9444/differences-between-ai-ml-dl-ds-cc899d6921bf

Machine Learning

Il machine learning (ML) studia metodi che possiamo applicare per construire applicazioni di IA che possono aiutare ad automatizzare diversi compiti, di solito di regressione o di predizione.

Si possono categorizzare in due classi in base al loro apprendimento:

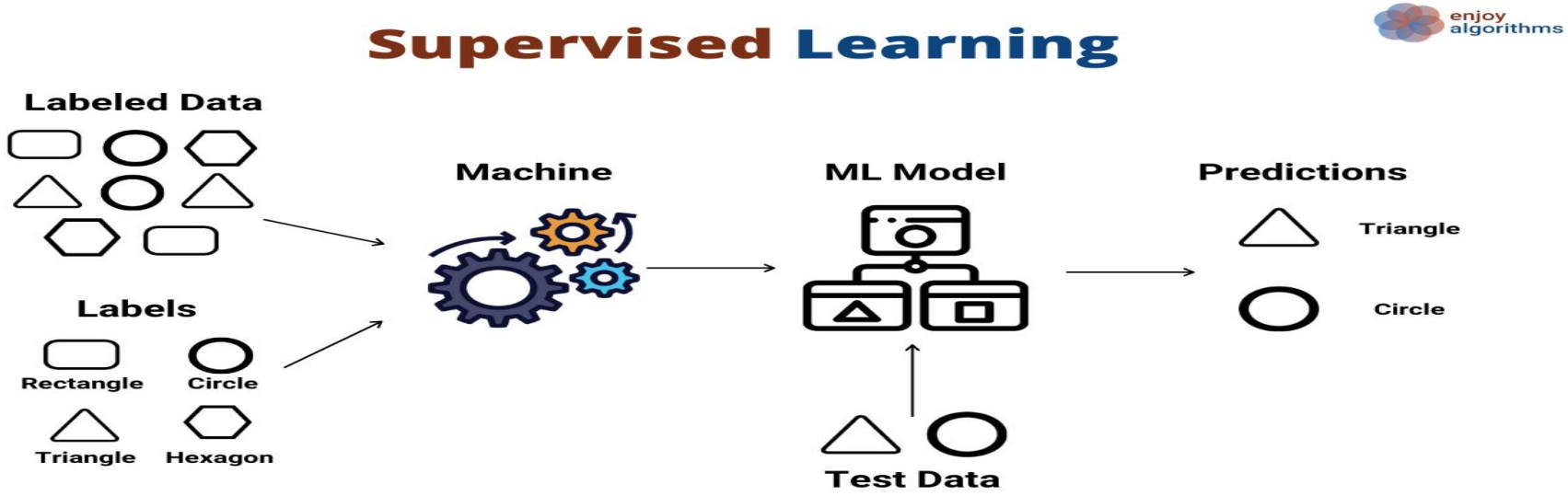
- Supervisionato (supervised): i dati sono forniti con delle “etichette” e l’algoritmo cerca di trovare una relazione tra esse e le variabili di input;

- Non supervisionato (unsupervised): i dati non sono etichettati e l’algoritmo cerca delle proprietà comuni ai campioni non conosciute in precedenza.

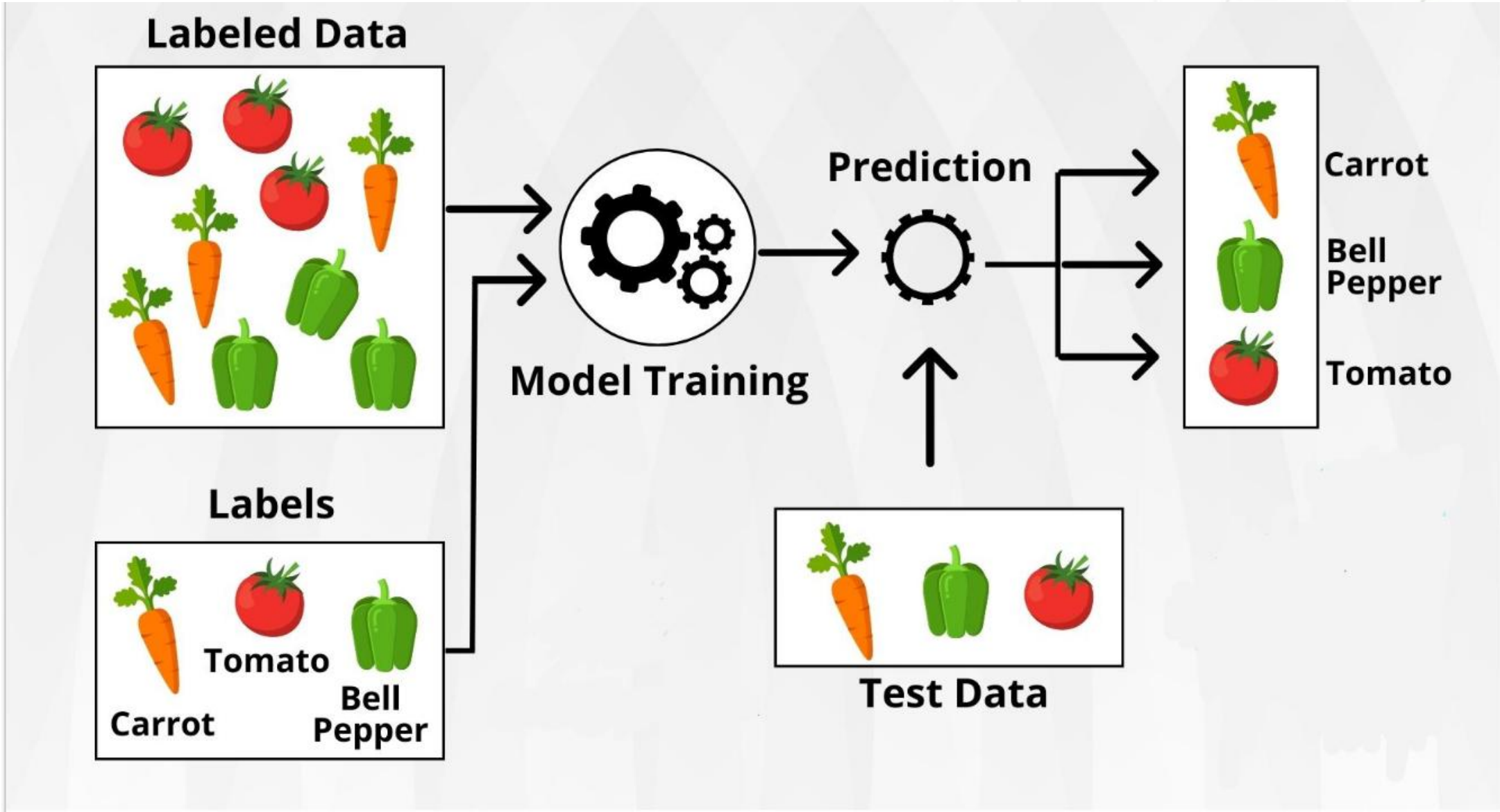
Supervised Machine Learning

Si lavora con dati etichettati e l'obiettivo è trovare una relazione tra le caratteristiche o features dei dati e la variabile target (**training**), in modo da effettuare predizioni su dati non etichettati (**inferencing**).
Gli algoritmi si dividono in 2 tipologie:

- Classificazione: la variabile target è **discreta** (una **classe**);
- Regressione: la variabile target è continua;



Supervised Machine Learning: esempio



Supervised Machine Learning

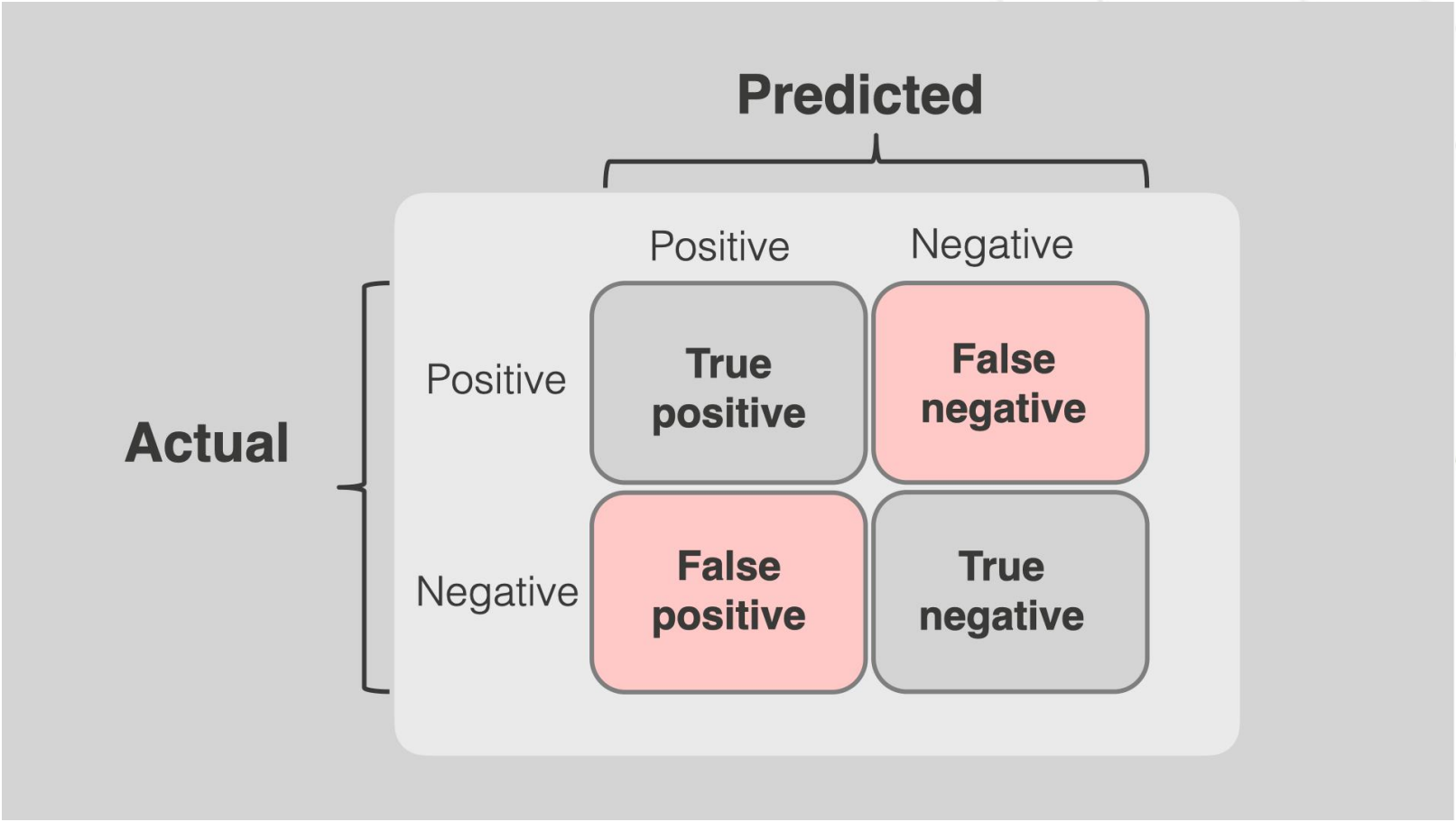
ITINERIS

Abbiamo bisogno di testare le performance del modello su dati non utilizzati durante il training. Di solito al momento della creazione del modello, si dividono i dati disponibili in due insiemi con una divisione 70%-30%.

- L’insieme più numeroso verrà usato per il training e viene chiamato **training set**. Questi dati verranno impiegati anche per ottimizzare il modello;
- L’altro verrà utilizzato al termine per valutare le performance del modello nella predizione a partire da dati sconosciuti. E’ chiamato **validation set**;

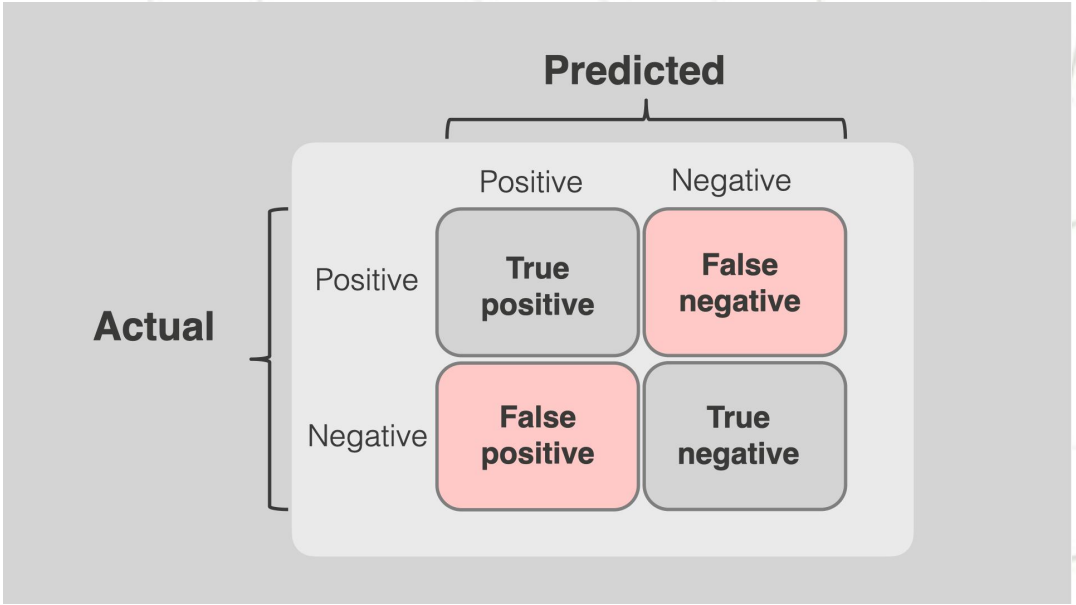
Supervised ML: valutare le performance

Ci sono diversi modi per valutare la qualità delle predizioni di un algoritmo di ML. Un esempio nel caso dei classificatori è la confusion matrix:



Supervised ML: valutare le performance

- Accuratezza: $\frac{TP+TN}{TP+TN+FP+FN}$;
- Precisione: $\frac{TP}{TP+FP}$;
- Specificità o Recall: $\frac{TP}{TP+FN}$
- Sensibilità o Sensitivity: $\frac{TN}{TN+FP}$
- F-score: $\frac{2 \cdot Prec \cdot Spec}{Prec + Spec}$



Supervised ML: Training/test split

Quando si divide il dataset in training e test set possono sorgere dei problemi:

- Quando si hanno pochi campioni, la divisione potrebbe lasciarci con troppi pochi dati per trainare il modello, ottenendo risultati molto variabili e/o scarsi;

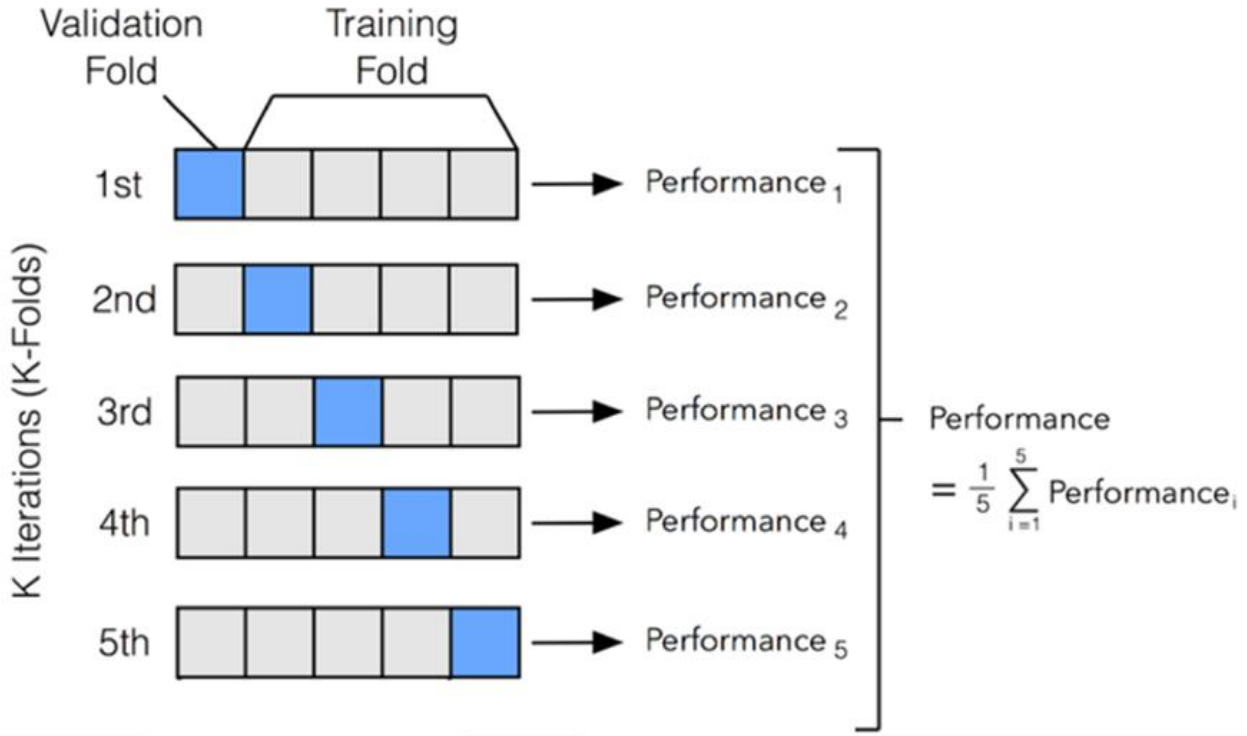
- La divisione, sebbene randomica, potrebbe generare dei bias, ovvero il modello effettua il training su un sottoinsieme di dati non rappresentativo dell'intero dataset. Ciò porta ad una cattiva performance nella predizione su nuovi dati;

Un modo per evitare queste situazioni è usare tecniche per utilizzare l'intero dataset, come la **K-fold cross validation**.

K-fold cross validation

- Si divide il dataset in k “pieghe” (le “folds”) in modo randomico. Uno di questi sottoinsiemi è usato per test, i restanti k-1 per il training;

- Il training viene effettuato per altre k volte, cambiano il fold per il test. Si ottengono cosi 10 modelli e 10 stime delle performance;



- Ogni campione verrà utilizzato k-1 volte per il training ed 1 volta per test;

- Le stime della qualità della predizione verranno usate per calcolare una media, indice anche della “robustezza” del modello;

Supervised ML: regressione

L'altro tipo di algoritmi di ML supervisionati sono i regressori, ovvero la predizione di variabili continue. I modelli sono trainati per approssimare una relazione tra i predittori (le variabili di input) e le variabili di target.

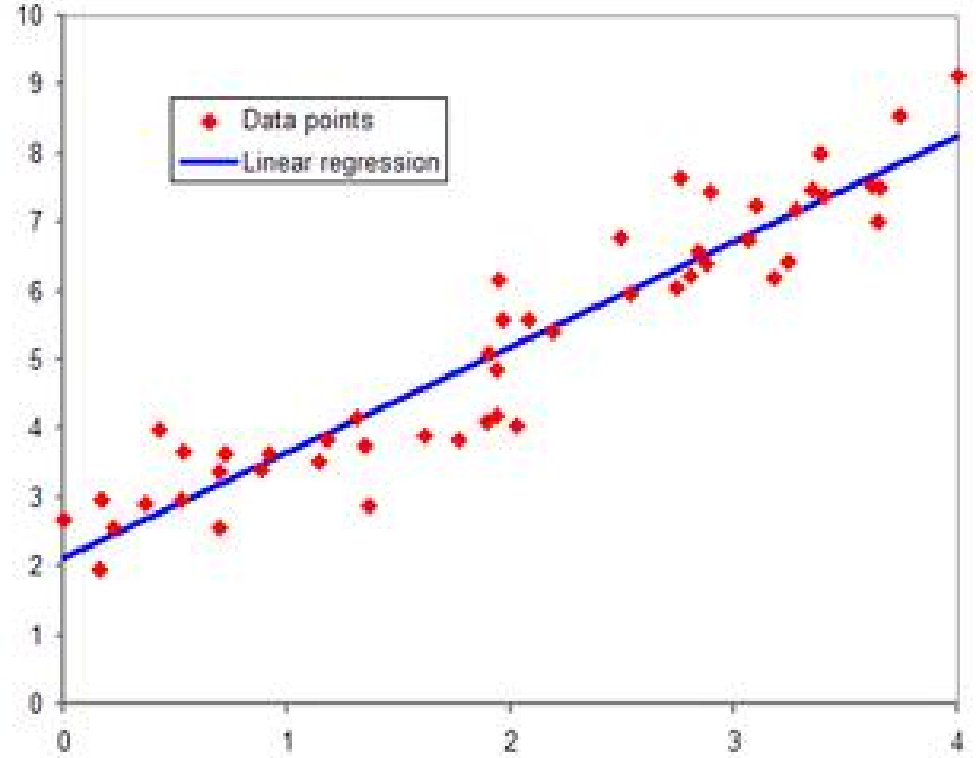
Ci sono varie metriche per valutare le performance del modello:

- $MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$, errore quadratico medio (mean squared error);
- $MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_I|^2}{n}$, errore assoluto medio (mean absolute error);
- $RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$, radice dell'errore quadratico medio (root MSE);

dove n è il numero di campioni, y_i il valore attuale della variabile target e $\hat{y}_i$ il valore predetto.

Supervised ML: regressione lineare

- Una singola variabile di input, anche chiamata variabile indipendente;
- Si cerca di approssimare la relazione tra variabile di input e la variabile target;



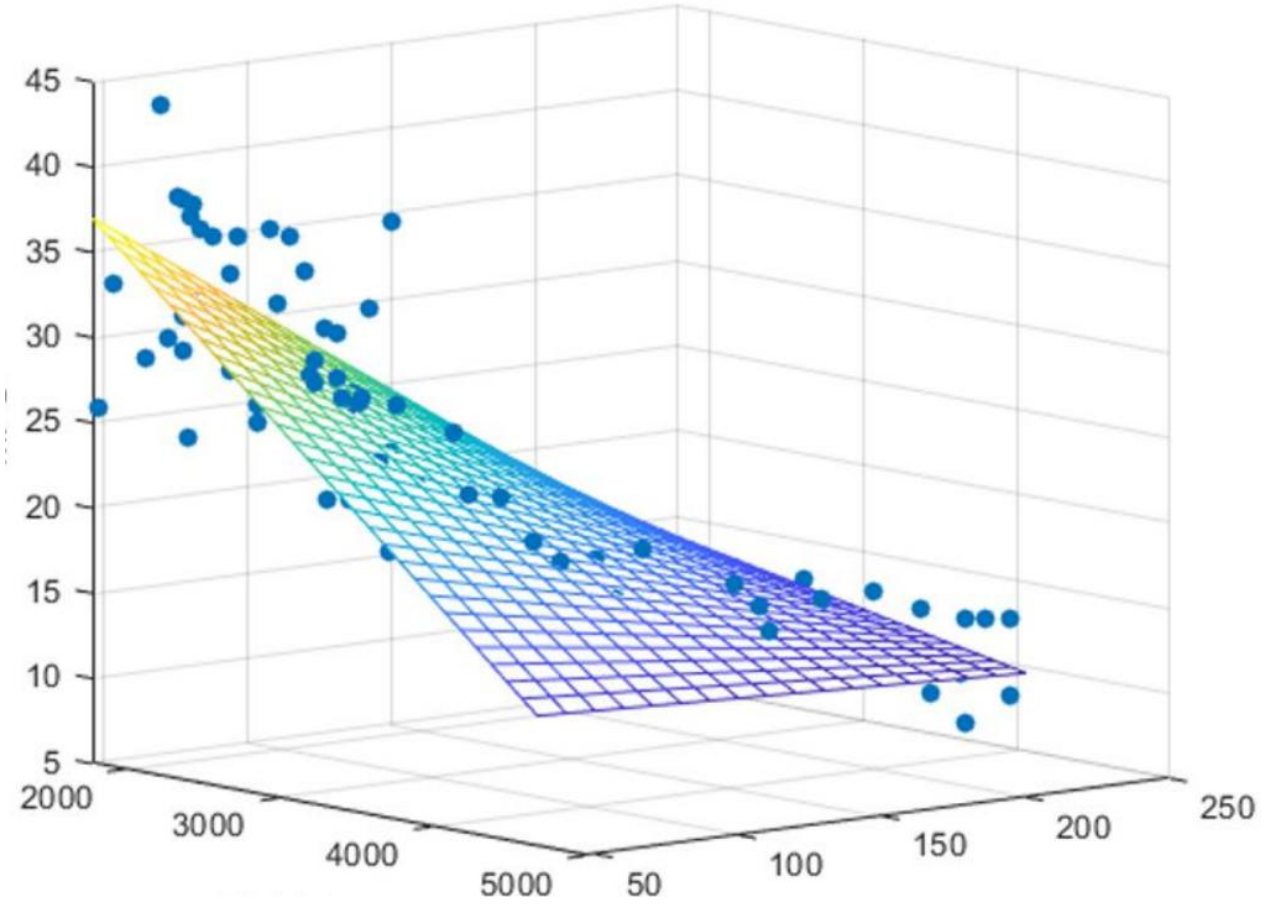
$$\hat{y} = a + bx$$

a intercetta
b coefficiente di regressione
x variabile di input
$\hat{y}$ predizione della variabile target

a e b vengono stimati durante il training

Supervised ML: regressione lineare multipla

- due o più variabili di input, anche chiamate variabili indipendenti;
- Si cerca di approssimare la relazione tra variabili di input e la variabile target;



$$\hat{y} = a + \sum_{i=0}^{n} b_i x_i$$

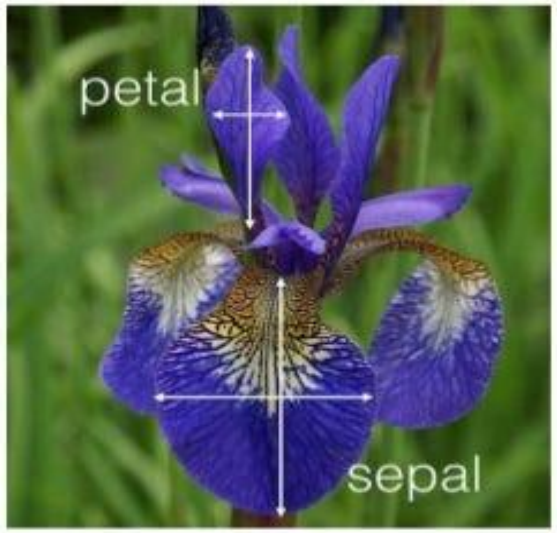
a intercetta
b_i coefficiente di regressione
x_i variabile di input
$\hat{y}$ predizione della variabile target

a e b vengono stimati durante il training

Iris dataset

Riutilizziamo l'iris dataset, utilizzando il pacchetto scikit-learn per recuperarlo. I dati sono forniti in un dataframe in cui le righe sono i **campioni** o **sample** e le colonne sono le **caratteristiche** misurate, anche dette **features**.

Supervised learning *classification* problem
(using the Iris flower data set)



Training / test data

Features				Labels
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	Iris setosa
4.9	3.0	1.4	0.2	Iris setosa
7.0	3.2	4.7	1.4	Iris versicolor
6.4	3.2	4.5	1.5	Iris versicolor
6.3	3.3	6.0	2.5	Iris virginica
5.8	3.3	6.0	2.5	Iris virginica

Iris dataset

Importiamo i packages necessari

```
from sklearn import datasets
import pandas as pd
import numpy as np
```

Carichiamo il dataset:

```
iris = datasets.load_iris
print(type(iris))
```

L'oggetto è di tipo bunch, simile a un dizionario. Salviamo le variabili di input e target separatamente

```
dataset = iris['data']
print("dataset :")
print(dataset[0])

target = iris['target']
print("target :")
print(target)
```

Lo 0 corrisponde alla specie Setosa, l'1 alla specie virginica, il 2 alla specie versicolor

Iris dataset

ITINERIS

Vogliamo trasferire tutto in un dataframe; abbiamo bisogno di un indice per le righe e i nomi per le colonne

```
n_samples = dataset.shape
print("numero campioni :")
print(n_samples)

index = np.array(range(0,n_samples[0]))
columns = iris['feature_names'] + ['Species']
```

concateniamo features e target e salviamolo in un dataframe

```
dataset_2 = np.concatenate((dataset,target.reshape(-1,1)), axis=1)
iris_df = pd.DataFrame(data=dataset_2, index=index, columns=columns)
iris_df.head()
```

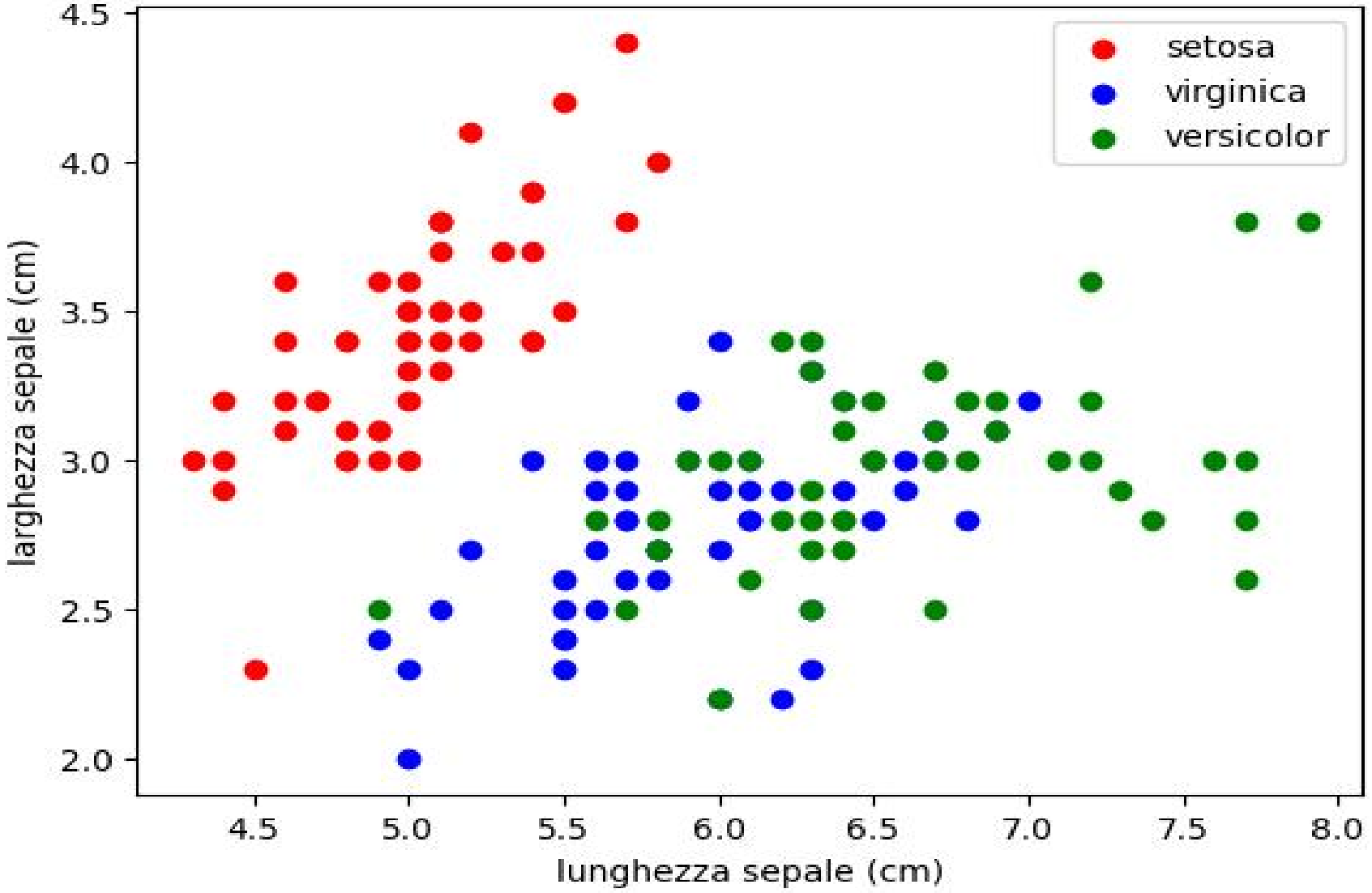
Iris dataset

Possiamo creare dei grafici con matplotlib e seaborn

```
import matplotlib.pyplot as plt
import seaborn as sns
```

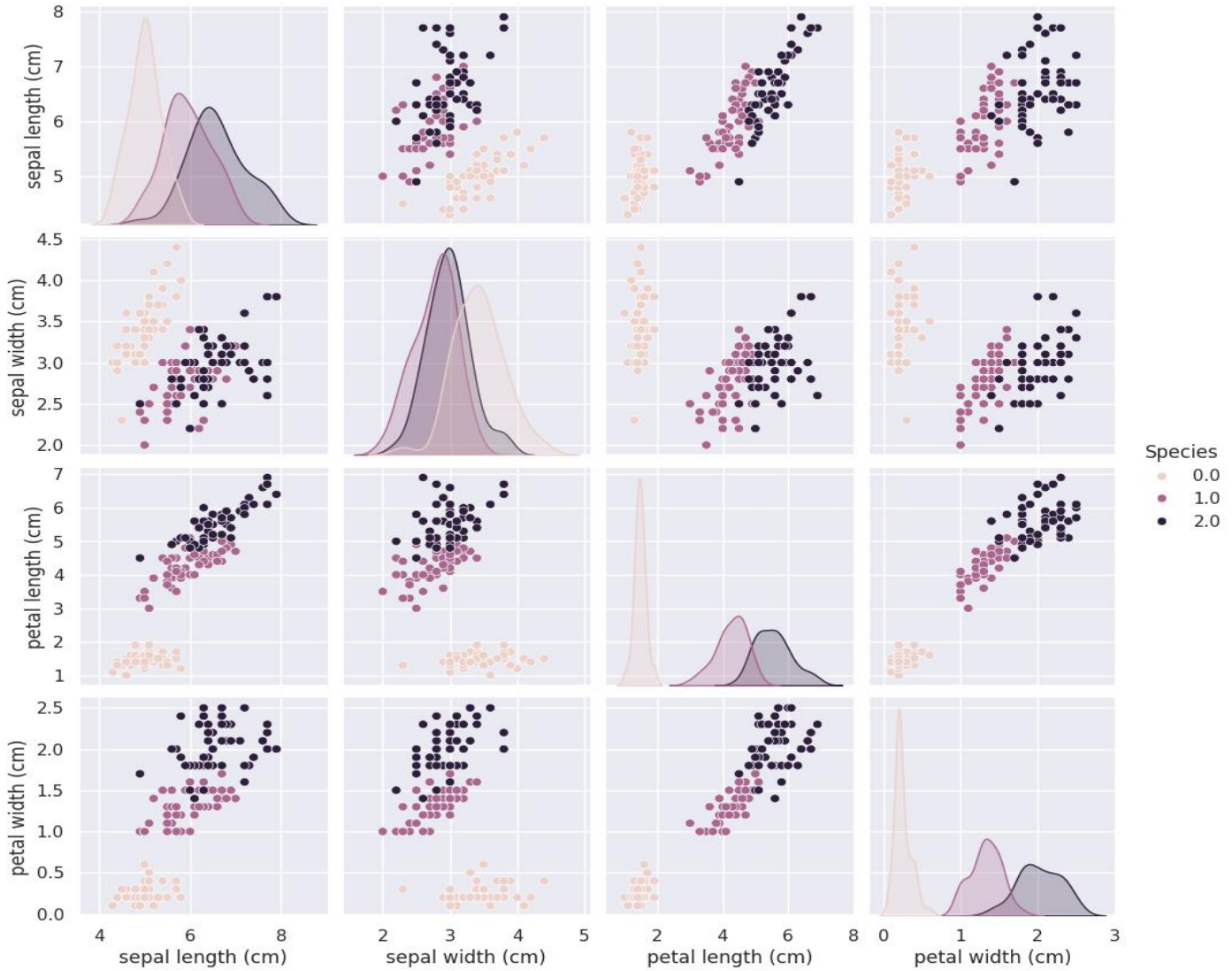
```
df_sepal = iris_df[['sepal length (cm)','sepal width (cm)', 'Species']]
setosa_sepal = df_sepal[df_sepal['Species'] == 0]
virginica_sepal = df_sepal[df_sepal['Species'] == 1]
versicolor_sepal = df_sepal[df_sepal['Species'] == 2]

plt.scatter(setosa_sepal['sepal length (cm)'], setosa_sepal['sepal width
(cm)'], color='red', label='setosa')
plt.scatter(virginica_sepal['sepal length (cm)'], virginica_sepal['sepal
width (cm)'], color='blue', label='virginica')
plt.scatter(versicolor_sepal['sepal length (cm)'], versicolor_sepal['sepal
width (cm)'], color='green', label='versicolor')
plt.xlabel('lunghezza sepale (cm)')
plt.ylabel('larghezza sepale (cm)')
plt.legend()
plt.show()
```

Iris dataset



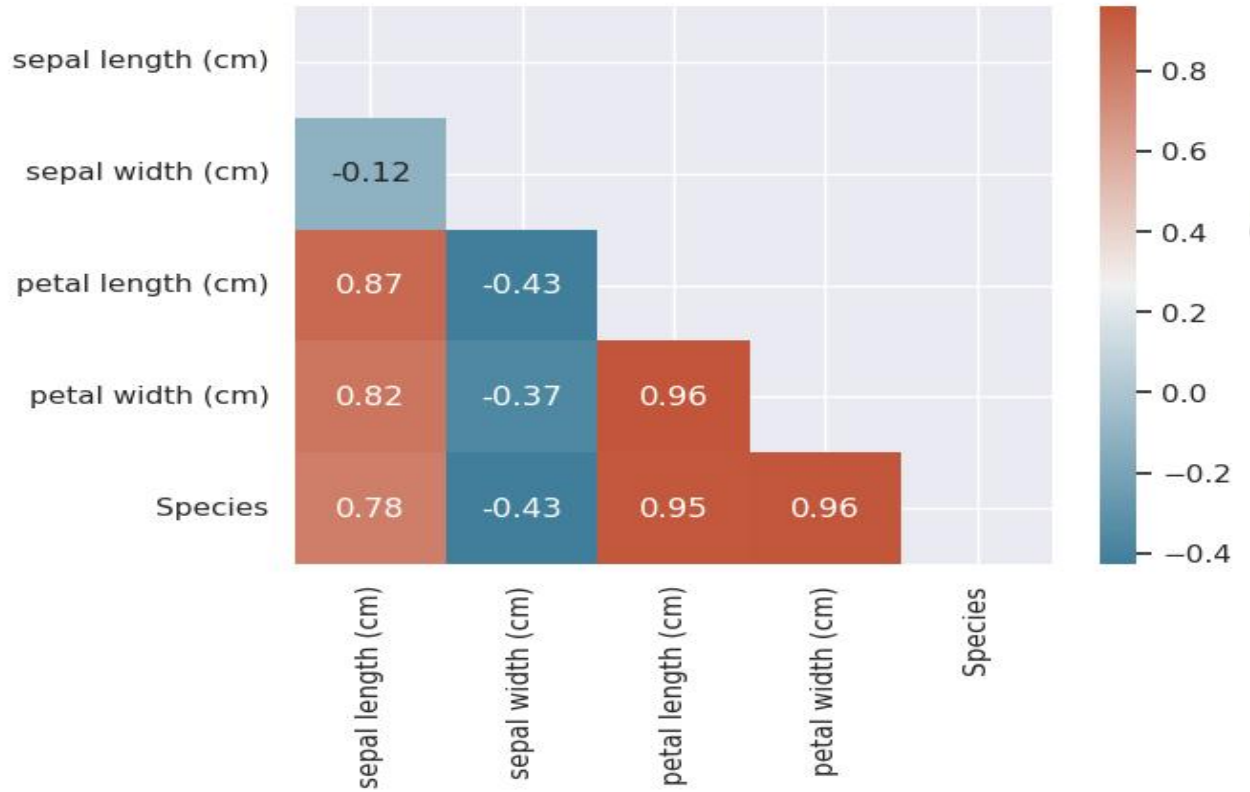
Iris dataset

```
sns.set()
sns.pairplot(iris_df, hue='Species')
```



Iris dataset: matrice di correlazione

```
corr = iris_df.corr()
cmap = sns.diverging_palette(h_neg=230, h_pos=20, as_cmap=True)

mask = np.triu(np.ones_like(corr, dtype=bool))

sns.heatmap(corr, annot=True, mask=mask, cmap=cmap)
```



Divisione set di training e di test

Sklearn mette a disposizione una funzione per dividere il dataset, train_test_split() che restituisce 4 output

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(dataset, target, random_state=0,test_size=0.33)
```

I 4 output sono le features e le variabili target per i due sottoinsiemi, in un ordine casuale definito dal parametro di input `random_state` e di dimensione stabilita dal parametro `test_size`.

```
print("Dimensioni X_train : {}".format(X_train.shape))
print("Dimensioni X_test : {}".format(X_test.shape))
print("Dimensioni y_train : {}".format(y_train.shape))
print("Dimensioni y_test : {}".format(y_test.shape))
```

K-cross fold validation

Sklearn offre anche una funzione per creare le k-cross fold a partire dai nostri dati:

```
from sklearn.model_selection import StratifiedKFold
```

```
SKF = StratifiedKFold(n_splits=5)
```

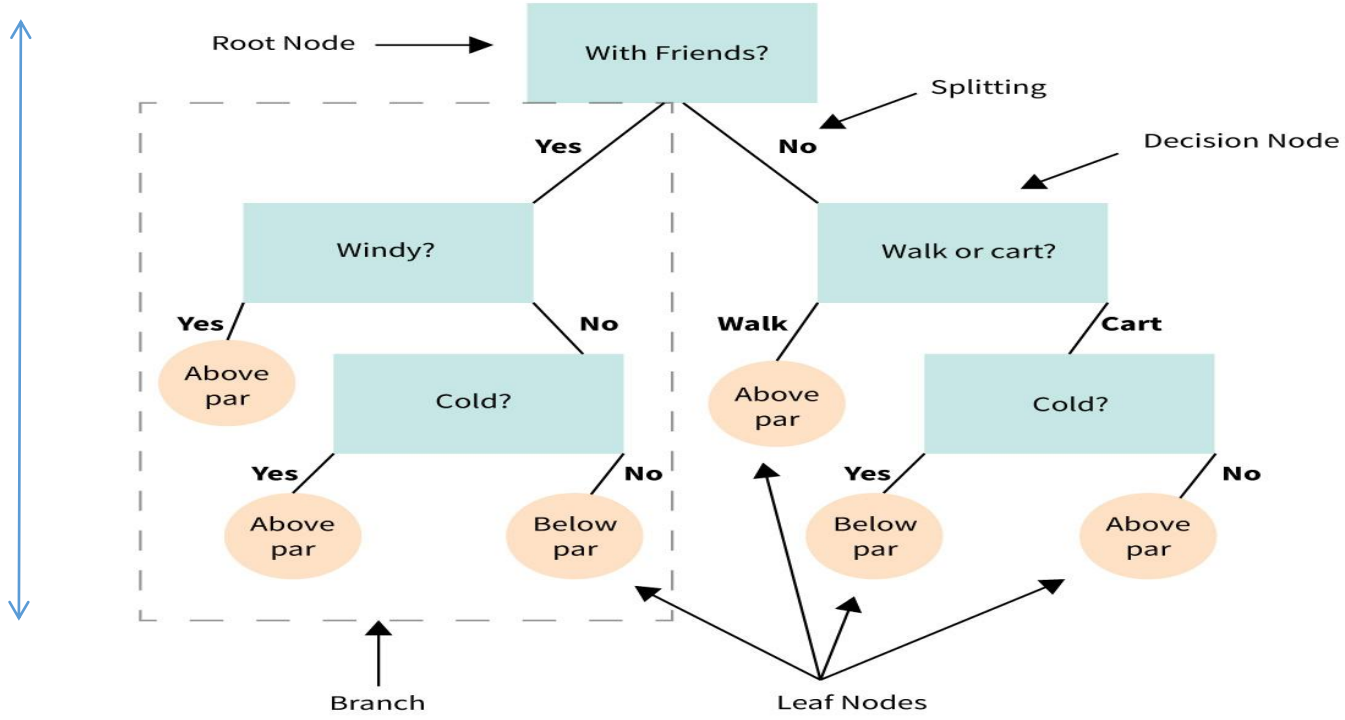
Viene creato un oggetto con un metodo split() che prende in input le features dei campioni e i valori target associati e effettua la divisione in 5 folds:

```
print(SKF.split(dataset, target))
for train_index, test_index in SKF.split(dataset, target):
    print("TRAIN:", train_index, " \n TEST:", test_index)
    X_train, X_test = dataset[train_index], dataset[test_index]
    y_train, y_test = target[train_index], target[test_index]
    print('')
```

Alberi di Decisione

- L’albero è una forma ricorrente nell’informatico, p.e. le directory;
- Gli alberi di decisione (decision trees, DT) sono dei diagrammi di flusso;
- I dati vengono separati sequenzialmente basandosi sulla soddisfazione di criteri o domande;

Altezza Albero



Alberi di Decisione: creazione

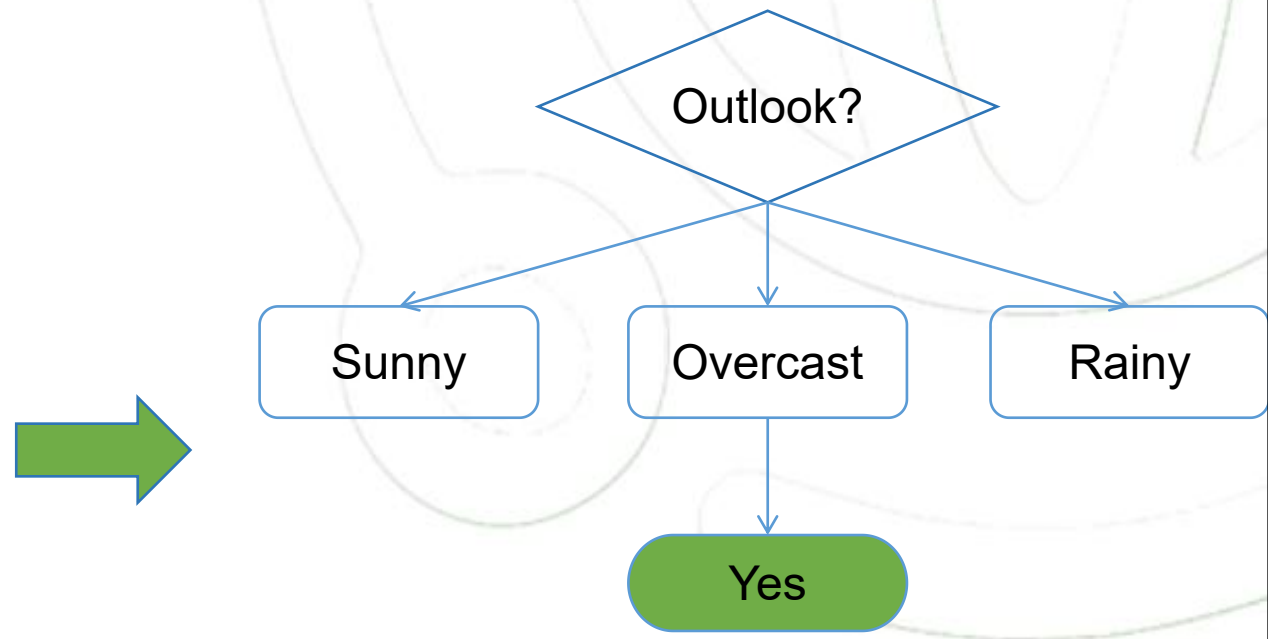
Abbiamo la seguente tabella che stabilisce se si giocherà o meno a tennis:

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

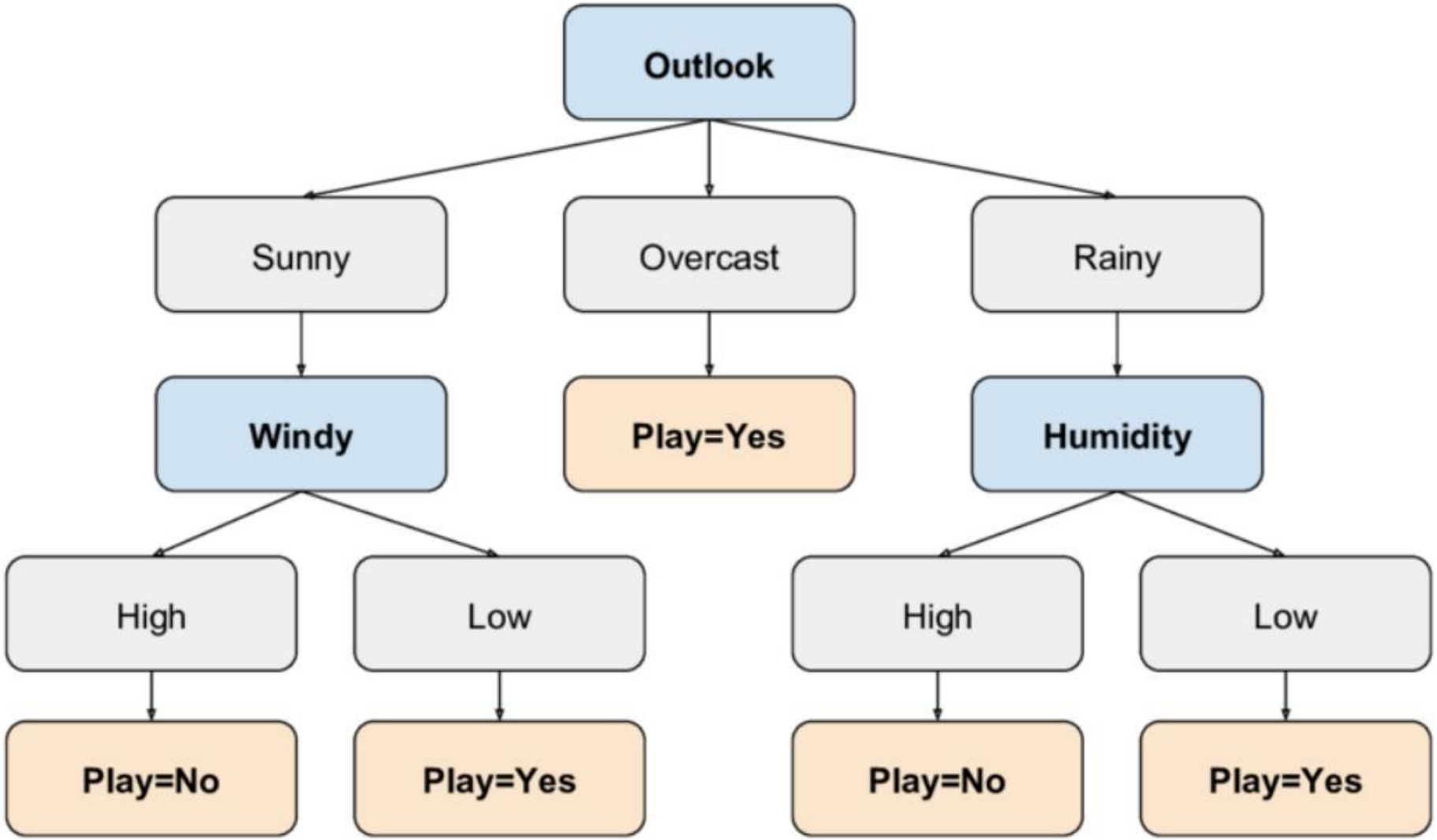
Alberi di Decisione: creazione

La prima variabile che consideriamo è l'outlook (ovvero il meteo). Per esempio, si può notare come quando è "overcast", si gioca sempre:

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No



Alberi di Decisione: creazione



Alberi di Decisione: creazione

- Abbiamo scelto arbitrariamente da quale variabile partire;
- A seconda della variabile scelta possiamo creare differenti alberi di decisione che arrivano tutti allo stesso output;
- Possono essere più o meno ridondanti;
- L’albero migliore è quello più corto!
- Come facciamo a determinare la feature che contiene più informazioni per la decisione dell’output finale (information gain)?

Alberi di Decisione: entropia

- Dato un insieme di classi (etichette) S e la loro frequenza p_i per ogni classe i, l'**entropia** è

$$E(S) = -\sum_{i=0}^{n} p_i \log_2 p_i$$

- Per esempio nel caso del tennis abbiamo 2 classi: si e no;
- In 9 casi l'output è "si", in 5 è "no":

$$E(S) = -\left(\frac{9}{14}\log_2\frac{9}{14} + \frac{5}{14}\log_2\frac{5}{14}\right) = 0.94$$

Alberi di Decisione: Information Gain

- Viene calcolata per ogni feature;
- Per una feature F che ha k valori ognuno con probabilità P_k:

$$IG(S,F) = \mathrm{E}(\mathrm{S}) - \sum_{j=1}^{k} P_k E\left(S_k\right)$$

- IG è la differenza tra l'entropia totale e l'entropia calcolata quando la feature ha un determinato valore;
- Conviene scegliere come variabile di scelta quella con IG più alto;

Alberi di decisione: IG - esempio

- Consideriamo la variabile “outlook”:

Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

$$E(S_{k=overcast}) = -\left(\frac{4}{4}\log_2\frac{4}{4}\right) = 0$$

$$E(S_{k=sunny}) = -\left(\frac{2}{5}\log_2\frac{2}{5} + \frac{3}{5}\log_2\frac{3}{5}\right) = 0.97$$

$$E(S_{k=rainy}) = -\left(\frac{3}{5}\log_2\frac{3}{5} + \frac{2}{5}\log_2\frac{2}{5}\right) = 0.97$$

$$IG(S, F = outlook) = E(S) - (P_{k=overcast}E(S_{k=overcast}) + P_{k=sunny}E(S_{k=sunny}) + P_{k=rainy}E(S_{k=rainy}) =$$

$$= 0.94 - \left(\frac{4}{14}\cdot 0 + \frac{5}{14}\cdot 0.97 + \frac{4}{14}\cdot 0.97\right) = 0.69$$

Alberi di decisione: IG - esempio

Effettuando lo stesso calcolo per tutte le features:

$$IG(S, F = \ outlook) = 0.69$$

$$IG(S, F = \ temperature) = 0.03$$

$$IG(S, F = \ humidity) = 0.215$$

$$IG(S, F = \ windy) = 0.07$$

Quindi si sceglie l'outlook come prima variabile.
Successivamente si ripete il procedimento per le restanti features fino a costruire l'intero albero.

Alberi di decisione in Python

Il pacchetto sklearn permette l'utilizzo di alberi in decisione. In particolare vedremo come usare un albero di decisione binario. Quindi:

- Ogni nodo può avere solo **due** nodi "figli": non si possono avere più di 3 esiti per ogni condizione;
- Le variabili categoriche devono essere convertite in **numeri** in modo da applicare i criteri di scelta tra i valori delle features;

Date queste condizioni l'albero ottenuto tramite sklearn sarà diverso dall'albero ottimale visto nelle slide precedenti.

Alberi di decisione in Python

Carichiamo e visualizziamo il dataset

```
import pandas as pd

dataset = pd.read_csv("play_tennis.txt", sep="\t")
print(dataset)
```

Trasformiamo le variabili categoriche in variabili numeriche tramite un encoding

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
dataset["play"] = le.fit_transform(dataset["play"])
dataset["outlook"] = le.fit_transform(dataset["outlook"])
dataset["temperature"] = le.fit_transform(dataset["temperature"])
dataset["humidity"] = le.fit_transform(dataset["humidity"])
dataset["windy"] = le.fit_transform(dataset["windy"])
print(dataset)
```

Alberi di decisione in Python

ITINERIS

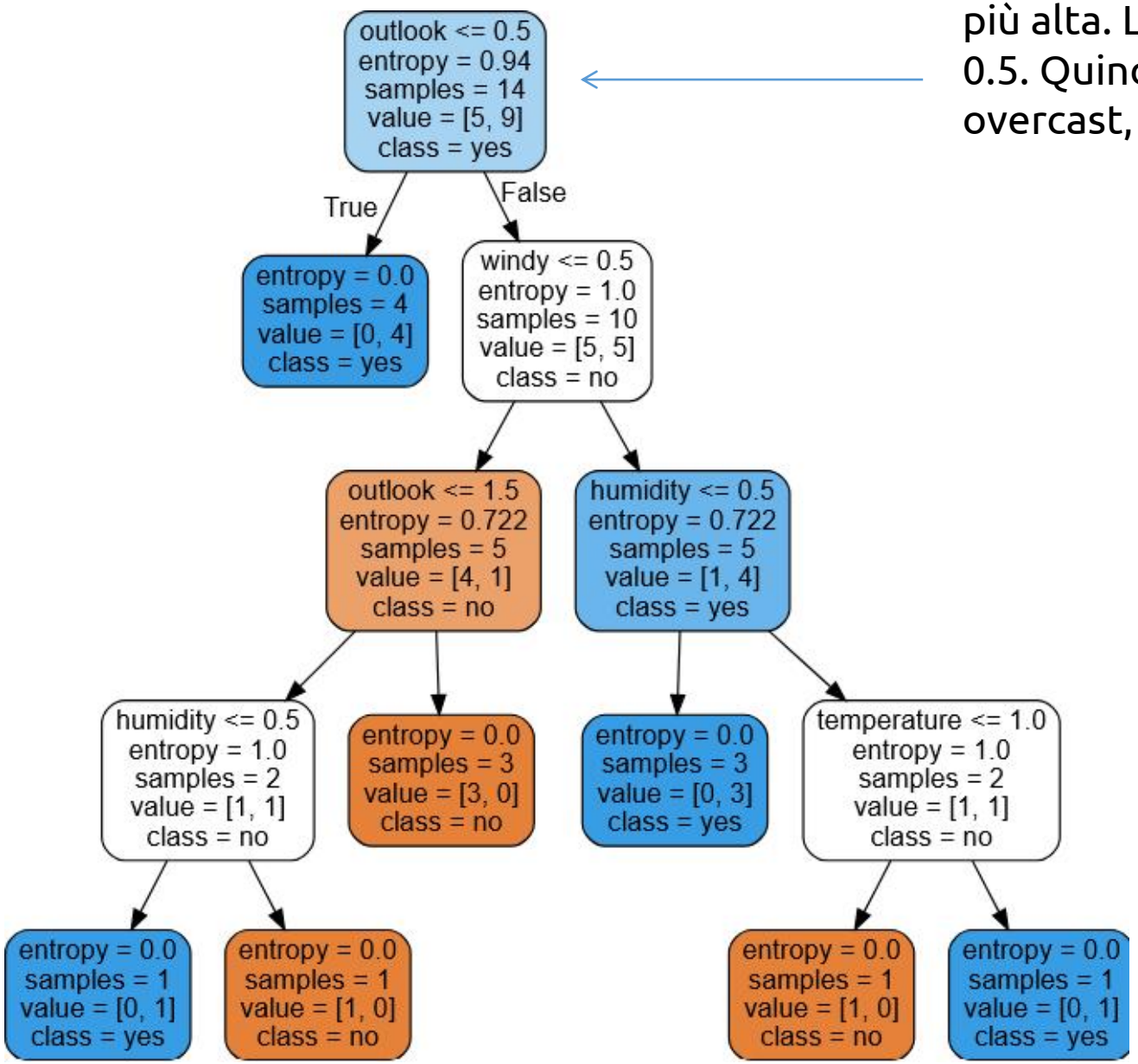
Creiamo e alleniamo l'albero di decisione

```
from sklearn.tree import DecisionTreeClassifier

x = dataset[["outlook", "temperature", "humidity", "windy"]]
y = dataset["play"]

dt = DecisionTreeClassifier(criterion="entropy", random_state=0)
dt.fit(x,y)
```

Possiamo anche visualizzare l'albero

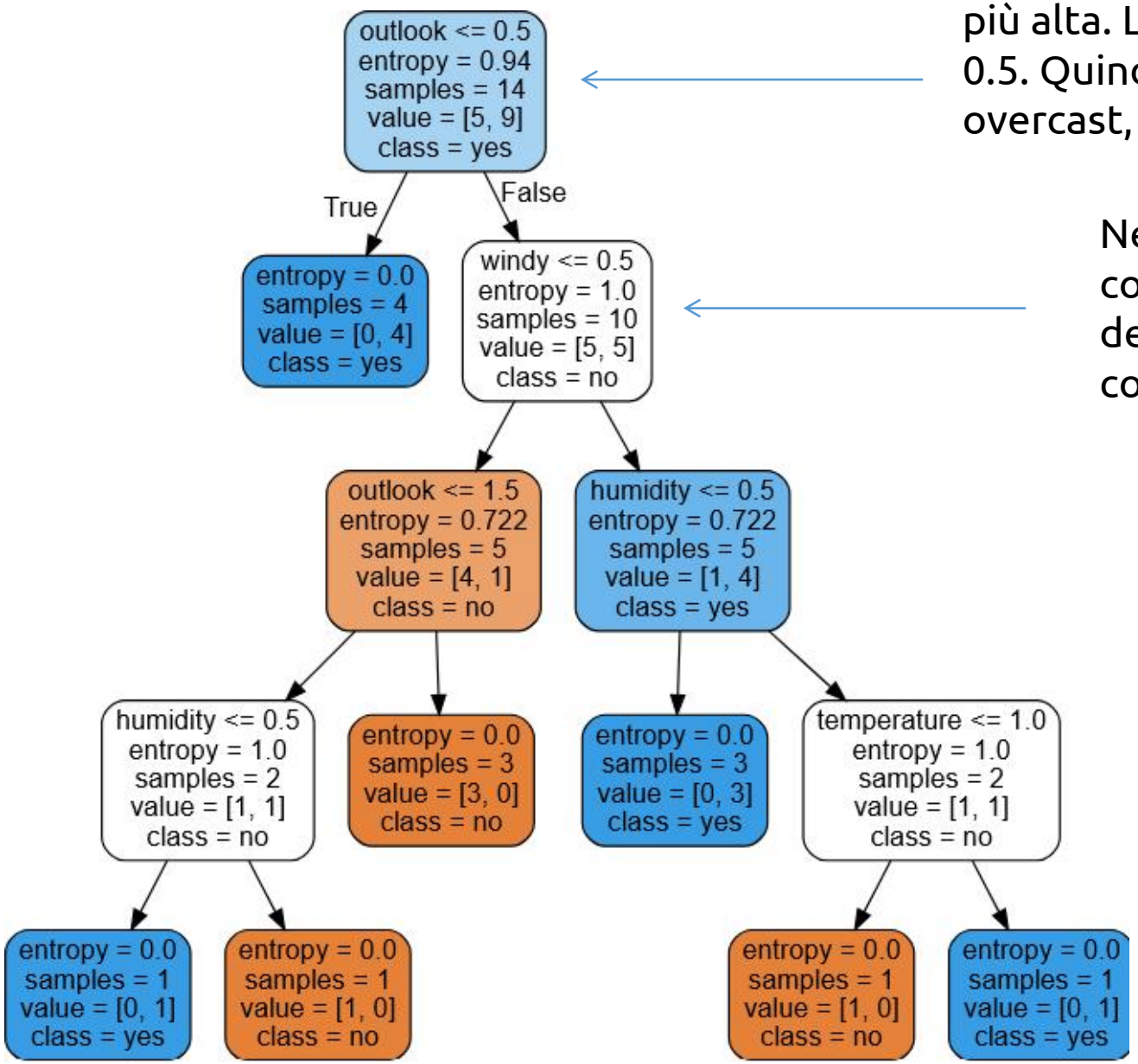
```
import graphviz
from sklearn import tree
dt_graph = tree.export_graphviz(dt, out_file=None,
                                feature_names=['outlook', 'temperature', 'humidity',
'windy'],
                                class_names=['no','yes'],
                                filled=True,
                                rounded=True)

graph = graphviz.Source(dt_graph, format="png")
graph
```

Alberi di decisione in Python



Outlook è la variabile con entropia più alta. La condizione è outlook <= 0.5. Quindi se outlook è 0 ovvero overcast, la classe è sempre sì.

Alberi di decisione in Python

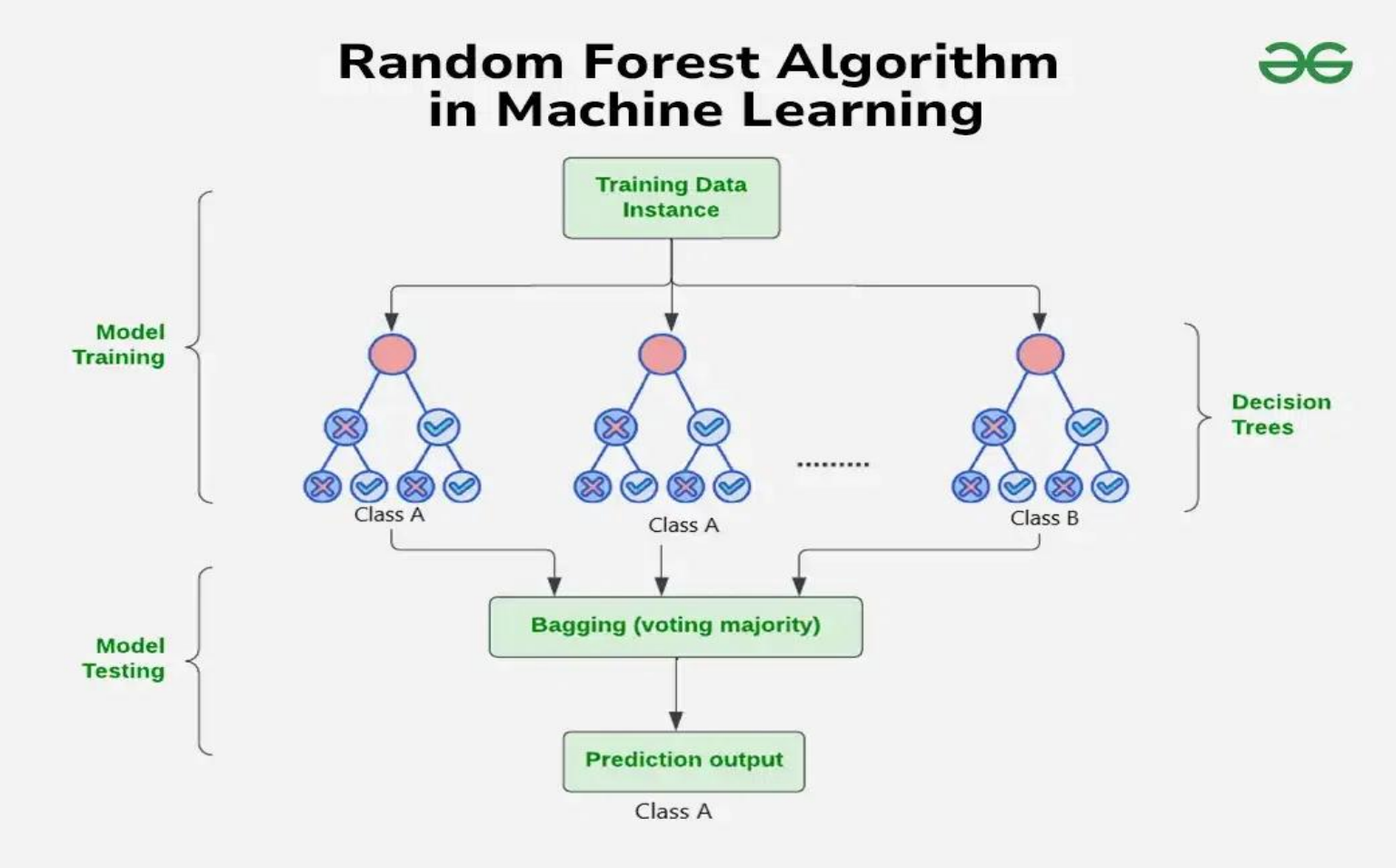


Outlook è la variabile con entropia più alta. La condizione è outlook <= 0.5. Quindi se outlook è 0 ovvero overcast, la classe è sempre sì.

Nell'altro nodo figlio sono contenute tutti gli altri valori dell'outlook, rainy e sunny. La nuova condizione è windy < 0.5.

Alberi di Decisione: RandomForest

Metodo di apprendimento d'insieme (ensemble method): più alberi di decisione e l'output finale viene scelto in base di un voto di maggioranza.



Alberi di Decisione: RandomForest

1. Seleziona un insieme casuale di *n* campioni;
2. Costruisce un albero di decisioni selezionando un insieme casuale di *d* features;
3. Ripete le operazioni per *k* volte (*k* è il numero di alberi desiderato);
4. Aggrega le predizioni scegliendo il valore che compare più volte come output;

RandomForest in Python

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
```

```
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target
df.head()
```

Separiamo features e variabile target

```
x = df.iloc[:, :-1].values
y = df['target']
```

Dividiamo in training e test set

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2, random_state=42)
```

RandomForest in Python

Scaliamo i dati per facilitare il training

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)
```

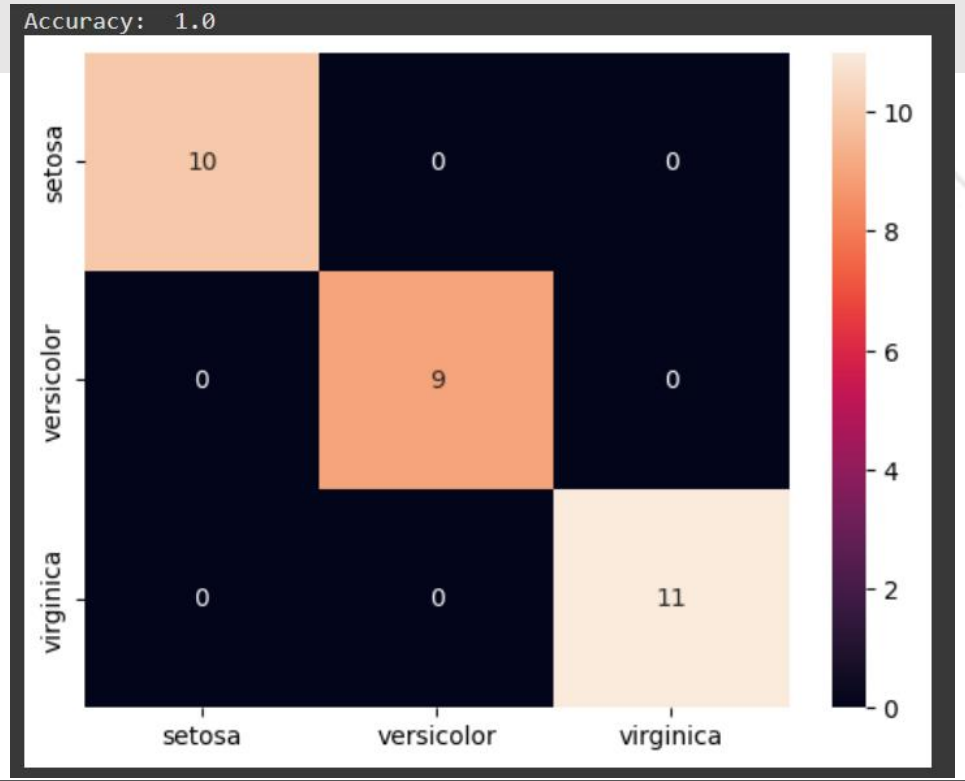
Scegliamo il classificatore e alleniamolo

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators=100,
random_state=42)
classifier.fit(x_train, y_train)
y_p = classifier.predict(x_test)
```

RandomForest in Python

Valutiamo le performance

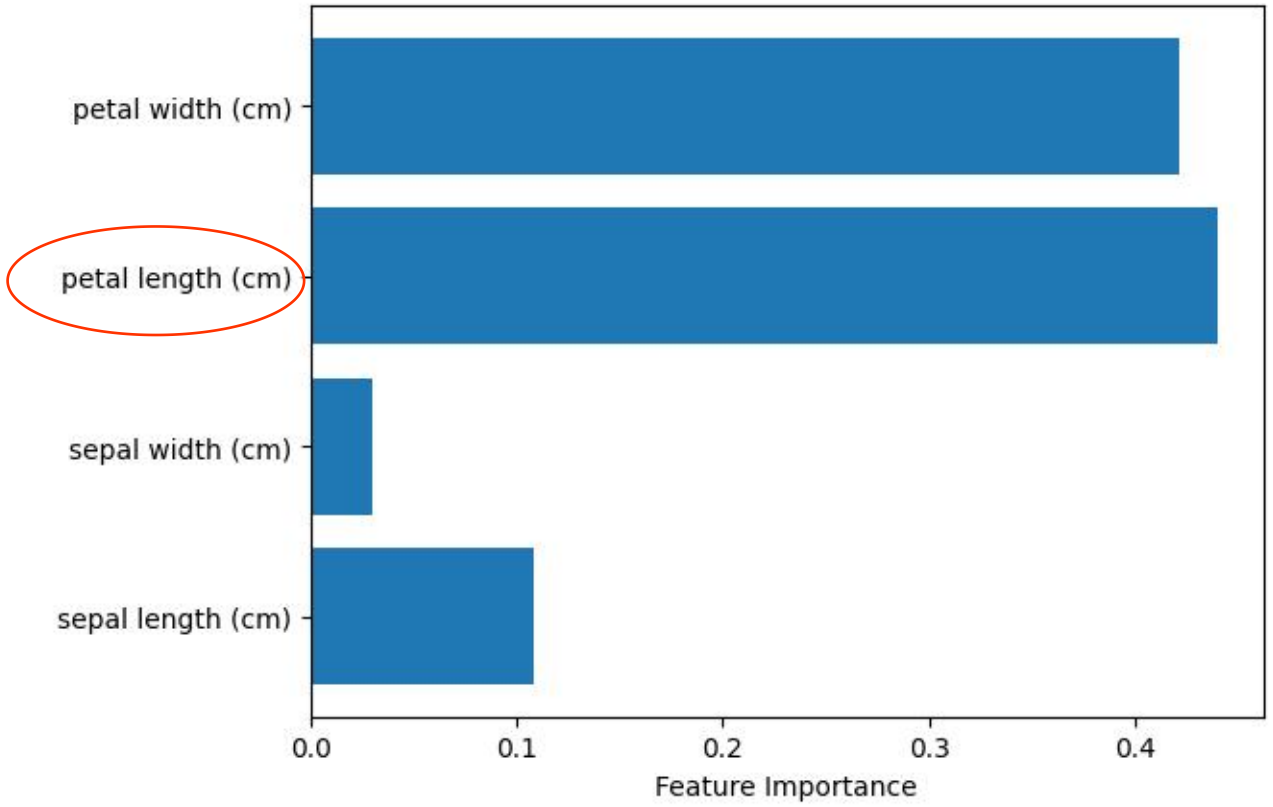
```
from sklearn.metrics import accuracy_score, confusion_matrix
acc = accuracy_score(y_test, y_p)
print('Accuracy: ', acc)
conf_mat = confusion_matrix(y_test, y_p)
sns.heatmap(conf_mat, annot=True, xticklabels=iris.target_names,
yticklabels=iris.target_names)
plt.show()
```



RandomForest in Python

Vediamo qual è la feature che influisce di più sulla classificazione

```
feat_imp = classifier.feature_importances_
plt.barh(iris.feature_names, feat_imp)
plt.xlabel('Feature Importance')
plt.show()
```



RandomForest in Python con CrossValidazione

Usiamo la CrossValidation per testare come il numero di alberi influisce sulle performance di un modello RandomForest

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('https://raw.githubusercontent.com/harjotspahwa/Car-Evaluation/refs/heads/master/car_evaluation.csv',header=None)
nomi_col = ['costo_iniz', 'costo_mant', 'porte', 'posti', 'bagagliaio', 'sicurezza', 'classe']
df.columns = nomi_col
print(df.info())
df.head()
```

Dataset in cui macchine sono valutate in base a costo e comfort e sono divise in 4 classi: inaccettabili, accettabili, buono, molto buono

	costo_iniz	costo_mant	porte	posti	bagagliaio	sicurezza	classe
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

RandomForest in Python con CrossValidazione

Usiamo la CrossValidation per testare come il numero di alberi influisce sulle performance di un modello RandomForest

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv('https://raw.githubusercontent.com/harjotspahwa/Car-Evaluation/refs/heads/master/car_evaluation.csv',header=None)
nomi_col = ['costo_iniz', 'costo_mant', 'porte', 'posti', 'bagagliaio', 'sicurezza', 'classe']
df.columns = nomi_col
print(df.info())
df.head()
```

Dataset in cui macchine sono valutate in base a costo e comfort e sono divise in 4 classi: inaccettabili, accettabili, buono, molto buono

	costo_iniz	costo_mant	porte	posti	bagagliaio	sicurezza	classe
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

RandomForest in Python con CrossValidazione

ITINERIS

trasformiamo le variabili categoriche in fattori

```
from sklearn.preprocessing import LabelEncoder
nomi_col = ['costo_iniz', 'costo_mant', 'porte', 'posti', 'bagagliaio',
'sicurezza', 'classe']
le = LabelEncoder()
df['costo_iniz'] = le.fit_transform(df['costo_iniz'])
df['costo_mant'] = le.fit_transform(df['costo_mant'])
df['porte'] = le.fit_transform(df['porte'])
df['posti'] = le.fit_transform(df['posti'])
df['bagagliaio'] = le.fit_transform(df['bagagliaio'])
df['sicurezza'] = le.fit_transform(df['sicurezza'])
print(df)
```

separiamo features e variabile target

```
x = df.drop(['classe'], axis=1)
y = df['classe']
```

RandomForest in Python con CrossValidazione

Generiamo delle fold per testare diversi parametri della RandomForest. Prepariamo anche delle liste per salvare i valori delle metriche per ogni valore dei parametri che vogliamo testare.

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
lista_acc = []
lista_prec = []
lista_recall = []
lista_f1 = []
lista_fold = []
```

Testiamo come cambiano le performance al variare del numero di alberi che compongono la RandomForest

```
val_par = [1,5,10,50,100]
```

RandomForest in Python con CrossValidazione

Effettuiamo il training dei modelli al variare del parametro n_estimators e calcoliamo le metriche sulla qualità delle performance per ogni fold

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
k=0
for train_index, test_index in skf.split(x,y):
  print("Fold: # ", k)
  print("n_alberi # ", val_par[k])
  x_train, x_test = x.iloc[train_index], x.iloc[test_index]
  y_train, y_test = y.iloc[train_index], y.iloc[test_index]
  model = RandomForestClassifier(criterion="entropy", random_state=42, n_estimators=val_par[k])
  model.fit(x_train, y_train)
  y_pred = model.predict(x_test)
  #print(y_pred.shape)
  acc = 100*accuracy_score(y_test, y_pred)
  recall = 100*recall_score(y_test, y_pred, average='weighted')
  prec = 100*precision_score(y_test, y_pred, average='weighted')
  f1 = 100*f1_score(y_test, y_pred, average='weighted')
  print("\taccuratezza: ", acc)
  print("\tspecificità: ", recall)
  print("\tprecisione: ", prec)
  print("\tf1 score: ", f1)
  lista_acc.append(acc)
  lista_prec.append(prec)
  lista_recall.append(recall)
  lista_f1.append(f1)
  lista_fold.append(k)
  k+=1
```

```
Fold: #  0
n_alberi #  1
        accuratezza:  90.17341040462428
        specificità:  90.17341040462428
        precisione:  91.12434148770473
        f1 score:  90.51404496420852
Fold: #  1
n_alberi #  5
        accuratezza:  95.08670520231213
        specificità:  95.08670520231213
        precisione:  95.1633133244405
        f1 score:  95.00102876027955
Fold: #  2
n_alberi #  10
        accuratezza:  95.66473988439307
        specificità:  95.66473988439307
        precisione:  96.17163183637172
        f1 score:  95.65840178480886
Fold: #  3
n_alberi #  50
        accuratezza:  97.10144927536231
        specificità:  97.10144927536231
        precisione:  97.23880662993194
        f1 score:  97.12781610238723
Fold: #  4
n_alberi #  100
        accuratezza:  97.68115942028986
        specificità:  97.68115942028986
        precisione:  97.68828700403898
        f1 score:  97.65939046937436
```

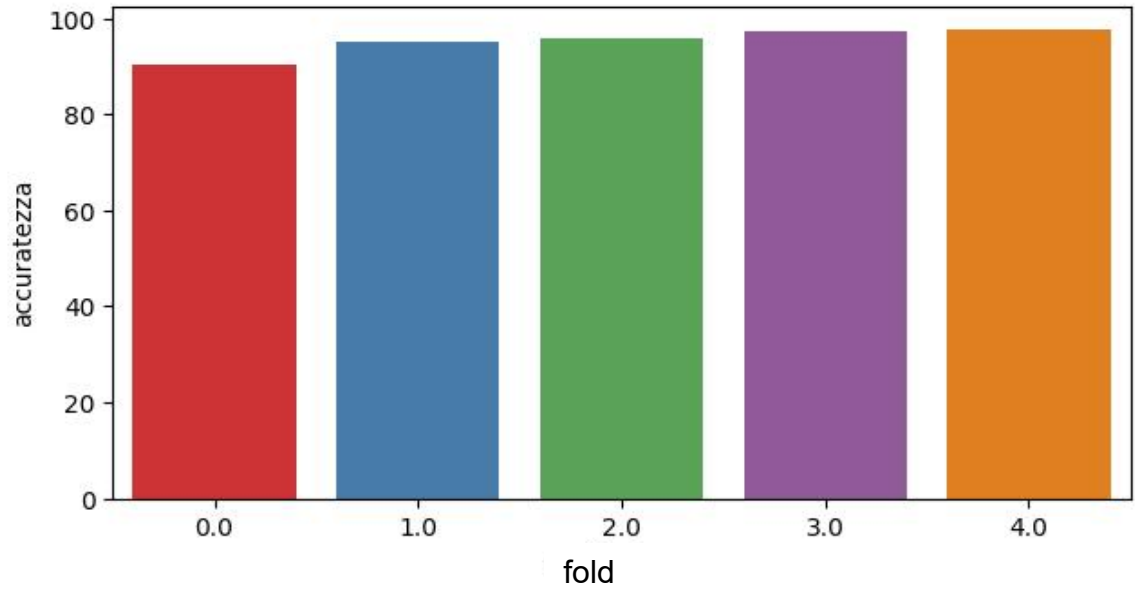
RandomForest in Python con CrossValidazione

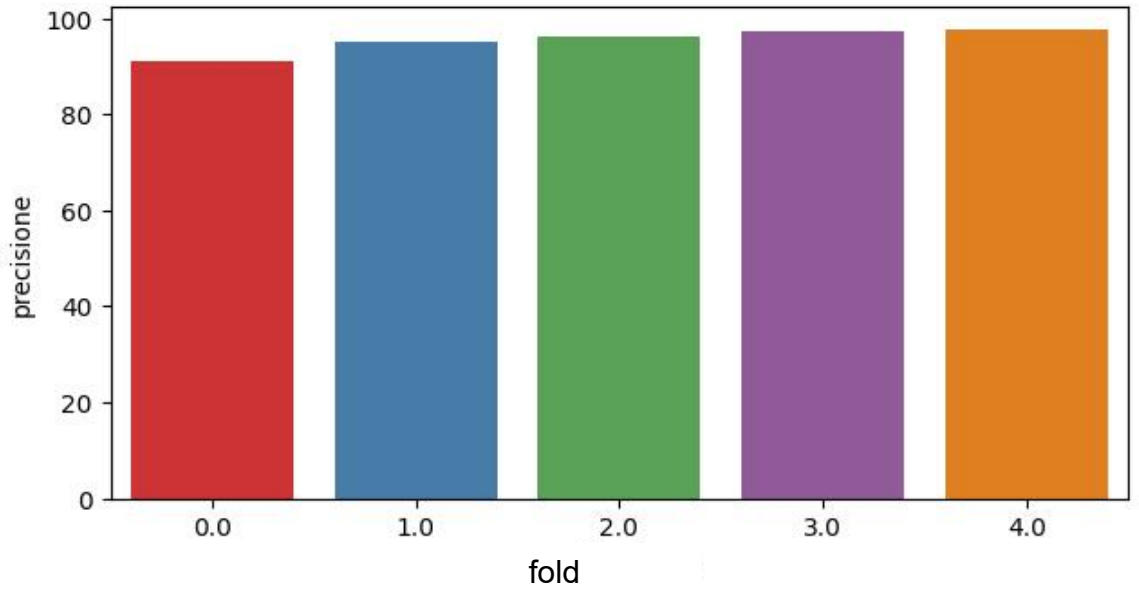
Visualizziamo i risultati con un grafico

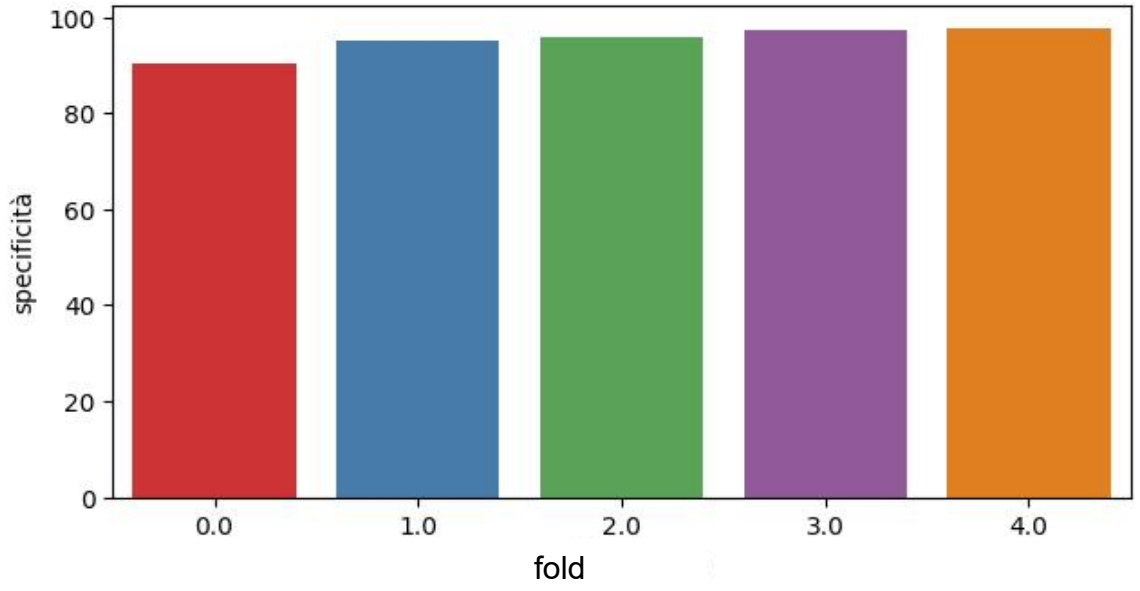
```
risultati = pd.DataFrame([lista_acc, lista_prec, lista_recall, lista_f1, lista_fold],
columns=[0,1,2,3,4], index=['accuratezza', 'precisione', 'specificità', 'f1 score', 'fold'])
risultati = risultati.T
fig, axes = plt.subplots(2,2,figsize=(16,8),sharey=False)

sns.barplot(data=risultati, ax=axes[0,0], x='fold', y='accuratezza', palette='Set1',
hue=[0,1,2,3,4], legend=False)
sns.barplot(data=risultati, ax=axes[0,1], x='fold', y='precisione', palette='Set1',
hue=[0,1,2,3,4], legend=False)
sns.barplot(data=risultati, ax=axes[1,0], x='fold', y='specificità', palette='Set1',
hue=[0,1,2,3,4], legend=False)
sns.barplot(data=risultati, ax=axes[1,1], x='fold', y='f1 score', palette='Set1',
hue=[0,1,2,3,4], legend=False)
```

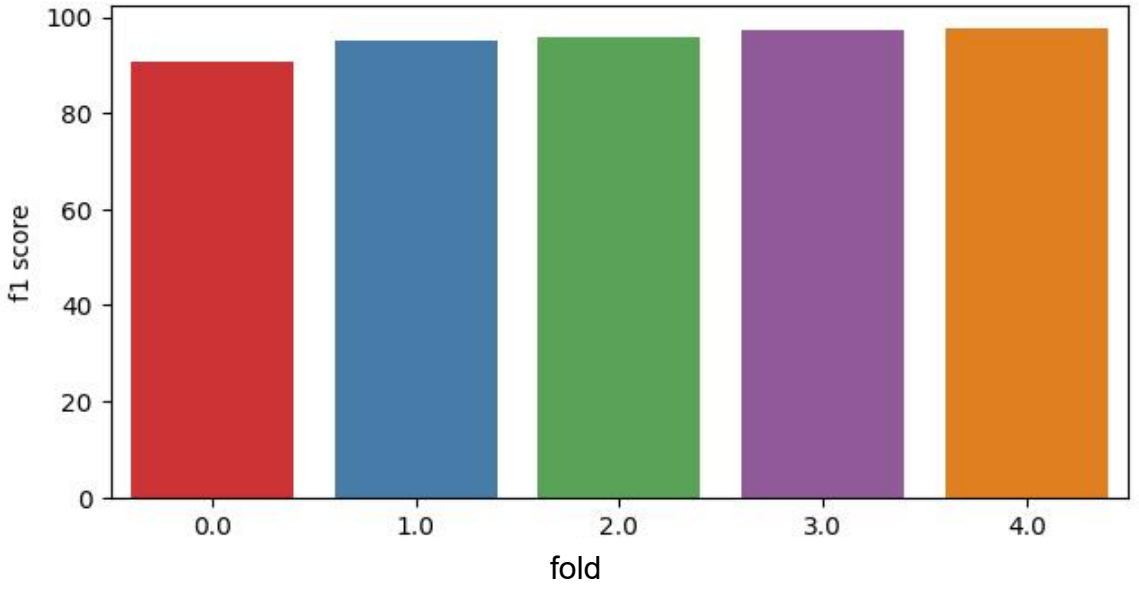
RandomForest in Python con CrossValidazione

Visualizziamo i risultati con un grafico







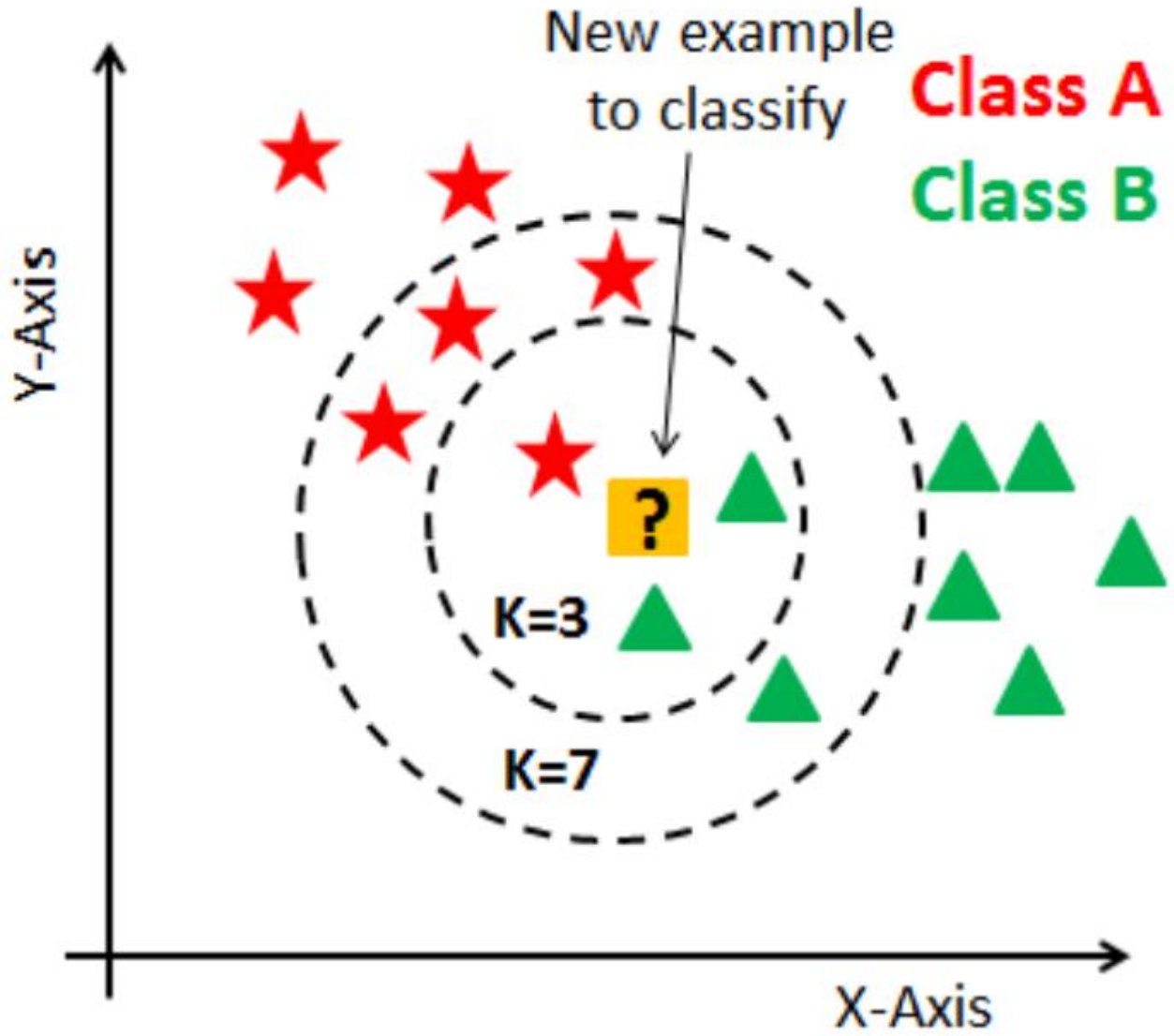


Algoritmo K-Nearest Neighbours (KNN)

L’algoritmo KNN è un algoritmo di classificazione di tipo “lazy learner”, apprendimento “pigro”:

- Non ricava una funzione che divide i dati di training, ma memorizza tutti i campioni in memorià e usa una metrica per stabilire la similarità;
- “Pigro” perchè non ha una fase specifica di training;
- Apprendimento senza uso di parametri -> Non fa assunzioni sui dati;
- Segue questi step:
  1. Si sceglie un numero k di “vicini” e una metrica di distanza;
  2. Per ogni campione da predirre si identificano i k vicini;
  3. Si assegna come classe quella maggiormente presente tra i k vicini

KNN


New example
to classify
Class A
Class B
per k=3
per k=7
Y-Axis
X-Axis
?
K=3
K=7

KNN

Dobbiamo scegliere una metrica per stabilire quali punti sono più vicini, spesso si usa la distanza euclidea:

$$D = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

In Python è molto semplice usare l'algoritmo KNN tramite sklearn:

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

KNN: Esercizio

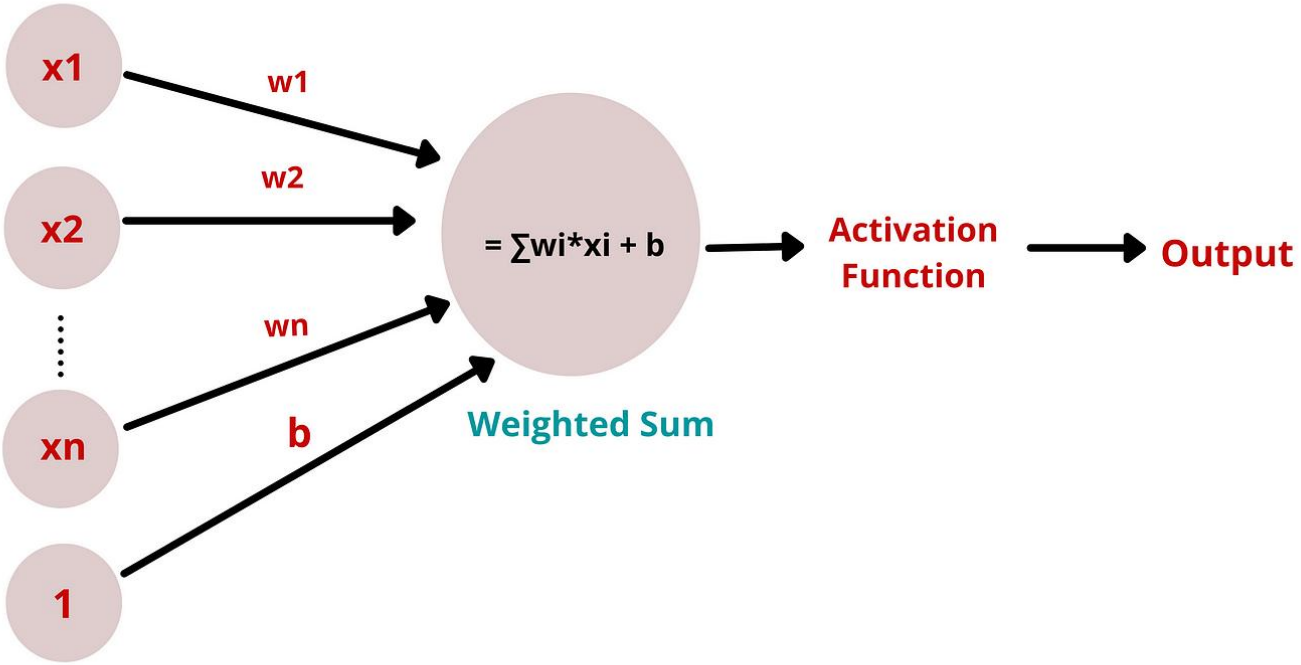
Applicare l'algoritmo KNN sul dataset iris. Usare la K-fold cross validation per testare diversi valori di k. Fornire dei grafici per la metrica che si vuole utilizzare per valutare la distanza tra i punti. Inoltre fornire uno o più grafici che aiutino a visualizzare i risultati al variare di k.

```
from sklearn.neighbors import KNeighborsClassifier
```

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

Reti Neurali

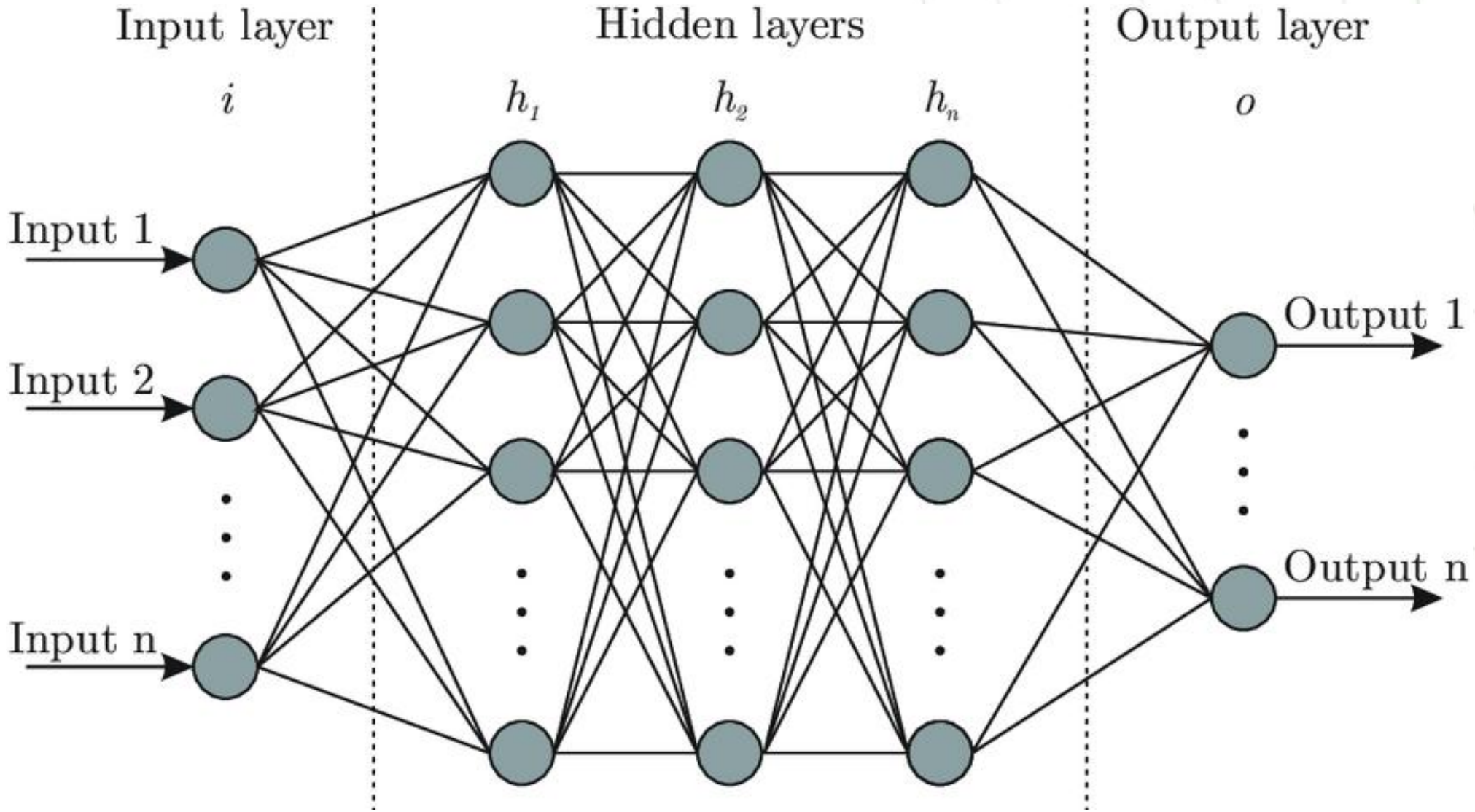
Le reti neurali (Neural Networks, NN) sono uno dei modelli di Deep Learning più usati, poichè altamente adattabili a diversi scopi.
L'unità più semplice delle NN è il **percettrone**:



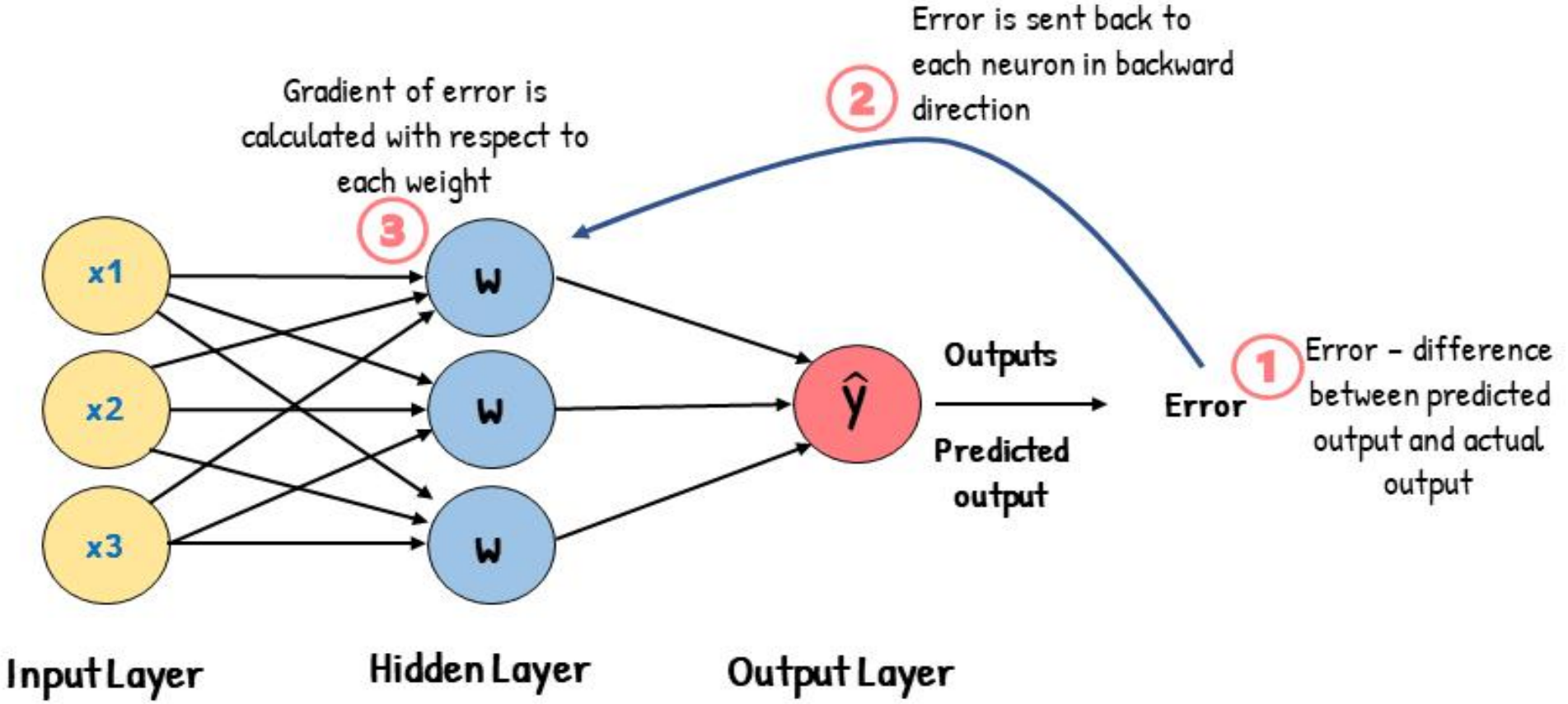
A Simple Neural Network

Reti Neurali

ITINERIS



Reti Neurali: apprendimento

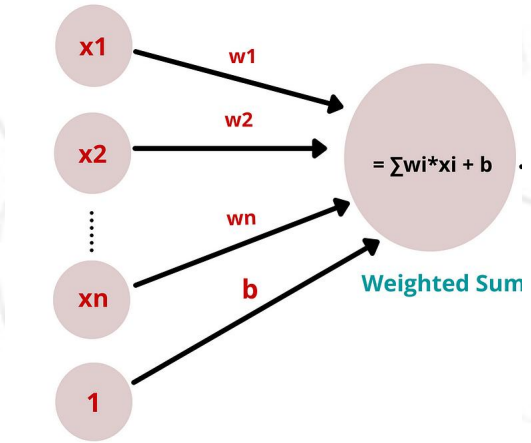


Aggiornamento dei pesi viene effettuato utilizzando un ottimizzatore, come lo Steepest Gradient Descent (SGD) o l'Adaptive Momnet Estimation (ADAM).

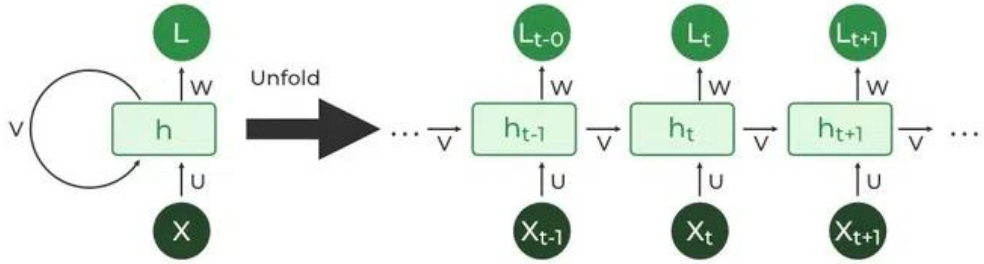
Reti Neurali: layers

Esistono molti tipi diversi di unità caratterizzate da che tipo di operazione performano sui dati di ingresso, alcuni dei più usati sono i seguenti:

- Lineari;



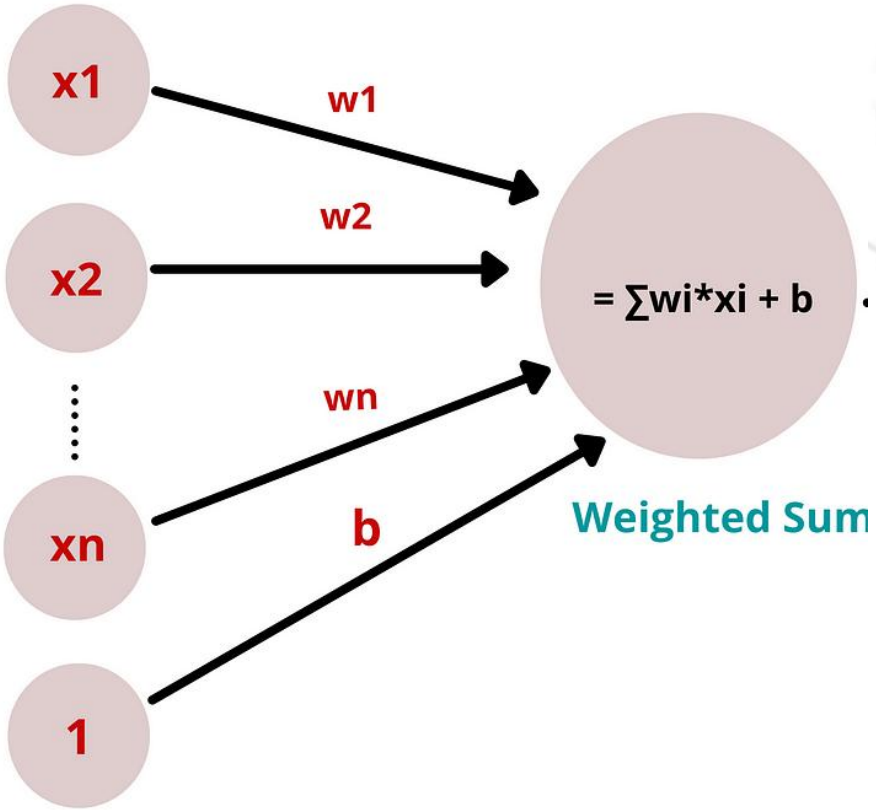
- Convoluzionali;



- Ricorsive;

Reti Neurali: layer lineare

Il layer più semplice, cerca una combinazione lineare delle componenti di input tale da approssimare il valore della variabile target. I componenti da ottimizzare è il vettore dei pesi *w* e eventualmente il bias *b*:



Reti Neurali: layer convoluzionale

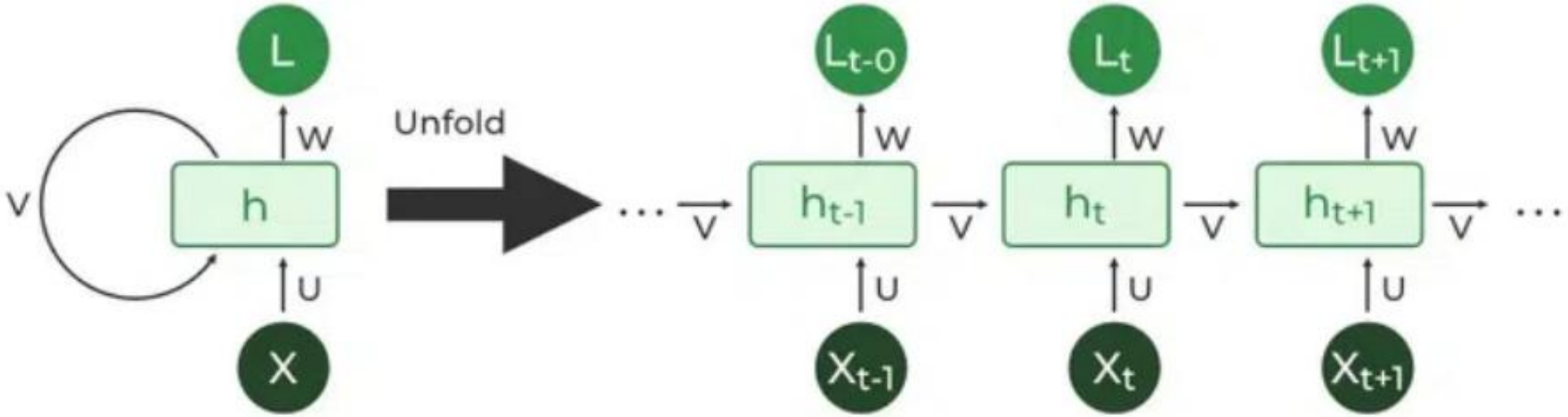
ITINERIS

Molto efficace nell'analisi di immagini e video, il layer convoluzionale raccoglie le informazioni nei dintorni del punto analizzato tramite dei filtri e le aggrega per poi produrre l'output desiderato:

Reti Neurali: layer ricorsivo

ITINERIS

Utile per analizzare dati di tipo temporale, i layer ricorsivi riutilizzano uno stesso neurone più volte, utilizzando come stato iniziale lo stato dei pesi del ciclo precedente. In questo modo riescono a ricavare informazioni dalle sequenze temporali:



Reti Neurali: funzioni di attivazione

Come abbiamo visto, i neuroni applicano principalmente semplici combinazioni lineari ai dati. Le funzioni di attivazione hanno principalmente tre scopi:

- Introdurre non linearità nella NN;
- Limitare l’intervallo dei valori possibili di output;
- “regolarizzare” il gradiente (vanishing o exploding gradient);

Reti Neurali in Python

ITINERIS

In Python esistono due framework principali per il Deep Learning:

TensorFlow

PyTorch

NN in Python: PyTorch

Utilizziamo un esempio riportato nella documentazione di PyTorch per identificare i componenti base di uno script di DL.

Fashion MNIST

Obiettivo: NN capace di identificare il tipo di vestiario

PyTorch: Dataset

ITINERIS

```
import torch
from torch.utils.data import Dataset
from torchvision import datasets
from torchvision.transforms import ToTensor
import matplotlib.pyplot as plt
```

```
train_data = datasets.FashionMNIST(
    root='data', #cartella dove salvare i dati
    train=True, #training set
    download=True,
    transform=ToTensor() #trasforma in tensore
)

test_data = datasets.FashionMNIST(
    root='data',
    train=False,
    download=True,
    transform=ToTensor()
)
```

PyTorch: Visualizzare i dati

ITINERIS

```
assoc_classi = {0: "T-Shirt", 1: "Trouser", 2: "Pullover", 3: "Dress", 4:
"Coat", 5: "Sandal", 6: "Shirt", 7: "Sneaker", 8: "Bag", 9: "Ankle Boot",}

figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols*rows+1):
  idx_campione = torch.randint(len(train_data), size=(1,)).item()
  img, label = train_data[idx_campione]
  figure.add_subplot(rows, cols, i)
  plt.title(assoc_classi[label])
  plt.axis('off')
  plt.imshow(img.squeeze(), cmap='gray')
plt.show()
```

PyTorch: DataLoader

Il dataloader si occupa di prelevare i sample dal dataset e permette di selezionare quanti campioni fornire alla rete durante ogni iterazione di training. Ciò è necessario perchè a seconda della dimensione del singolo campione, bisogna valutarne quanti di essi possono entrare in memoria (p.e. la RAM di una GPU). Inoltre si occupa anche di mescolare i campioni durante la fase di training per evitare che un ordine specifico influenzi il training.

```
from torch.utils.data import DataLoader

train_dataloader = DataLoader(train_data,
                              batch_size=64, #dimensione della batch
                              shuffle=True #mischiare i sample
                              )

test_dataloader = DataLoader(test_data, batch_size=64, shuffle=False)
```

PyTorch: DataLoader

```
train_features, train_labels = next(iter(train_dataloader))
print(f"Dimensione feature del batch: {train_features.size()}")
print(f"Dimensione etichette del batch: {train_labels.size()}")
img = train_features[0].squeeze()
label = train_labels[0]
plt.imshow(img, cmap="gray")
plt.show()
print(f"Classe: {label}")
```

Notiamo che abbiamo un tensore composto da 64 campioni di dimensioni (1,28,28) ovvero un solo canale composto da 28x28 pixel.
Ogni volta che eseguiamo la cella, l'immagine cambierà perchè abbiamo abilitato lo shuffle per il dataloader dei dati di test.

PyTorch: Costruzione della rete

ITINERIS

```
import torch
from torch import nn
```

```
class ReteNeurale(nn.Module):
  def __init__(self):
    super().__init__()
    self.flatten = nn.Flatten()
    self.linear_relu_stack = nn.Sequential(
        nn.Linear(28*28, 512), #immagini 28x28 pixel
        nn.ReLU(),
        nn.Linear(512,512),
        nn.ReLU(),
        nn.Linear(512,10) #output probabilità di appartenere alle 10 classi
    )

  def forward(self, x):
    x = self.flatten(x)
    logits = self.linear_relu_stack(x)
    return logits
```

PyTorch: Costruzione della rete

ITINERIS

```
device = torch.accelerator.current_accelerator().type if
torch.accelerator.is_available() else "cpu"
print(f"sto usando {device}")
```

```
model = ReteNeurale().to(device)
print(model)
```

```
ReteNeurale(
  (flatten): Flatten(start_dim=1, end_dim=-1)
  (linear_relu_stack): Sequential(
    (0): Linear(in_features=784, out_features=512, bias=True)
    (1): ReLU()
    (2): Linear(in_features=512, out_features=512, bias=True)
    (3): ReLU()
    (4): Linear(in_features=512, out_features=10, bias=True)
  )
)
```

PyTorch: Parametri e Iperparametri

```
print(f"struttura del modello: {model}\n\n")

for name, param in model.named_parameters():
  print(f"Layer: {name} | dimensione: {param.size()} | Valori: {param[:2]}
\n")
```


```
print(f"struttura del modello: {model}\n\n")

for name, param in model.named_parameters():
  print(f"Layer: {name} | dimensione: {param.size()} | Valori: {param[:2]}
\n")
```


Definiamo ottimizzatore e metrica di errore con relativi parametri

```
funz_loss = nn.CrossEntropyLoss()
learning_rate = 1e-3
ottimizzatore = torch.optim.SGD(model.parameters(), lr=learning_rate)
batch_size = 64
epochs = 5
```

PyTorch: Training Loop

ITINERIS

```
def train_loop(dataloader, model, funz_loss, ottimizzatore, batch_size):
  dim = len(dataloader.dataset)
  model.train() #imposta il modello in training mode
  for batch, (x, y) in enumerate(dataloader):
    x,y = x.to(device), y.to(device)
    pred = model(x)
    loss = funz_loss(pred,y)
    loss.backward() #Calcolo del gradiente
    ottimizzatore.step() #assegnazione nuovi pesi
    ottimizzatore.zero_grad() #azzero i gradienti
    if (batch+1)%100 == 0:
      loss = loss.item()
      img_att = batch*batch_size + len(x)
      print(f"loss: {loss:>7f} [{img_att:>5d}/{dim:>5d}]")
```

PyTorch: Test Loop

ITINERIS

```
def test_loop(dataloader, model, funz_loss):
  model.eval() #imposta il modello in evaluation mode
  dim = len(dataloader.dataset)
  num_batches = len(dataloader)
  test_loss = 0
  corretti = 0
  with torch.no_grad(): #non c'è bisogno di calcolare i gradienti
    for x,y in dataloader:
      x,y = x.to(device), y.to(device)
      pred = model(x)
      test_loss += funz_loss(pred,y).item()
      corretti += (pred.argmax(1) == y).type(torch.float).sum().item()

  test_loss /= num_batches
  corretti /= dim
  print(f"Errore valutazione: \n Accuracy: {(100*corretti):>0.1f}%, errore
medio: {test_loss:>8f} \n")
```

PyTorch: Training e validazione

```
for t in range(10):
    print(f"Epoch {t+1}\n-------------------------------")
    train_loop(train_dataloader, model, funz_loss, ottimizzatore, batch_size)
    test_loop(test_dataloader, model, funz_loss)
print("Fatto!")
```

Output ultimo ciclo:

```
Epoch 10
-------------------------------
loss: 0.593357 [ 6400/60000]
loss: 0.697068 [12800/60000]
loss: 0.798004 [19200/60000]
loss: 0.684591 [25600/60000]
loss: 0.667704 [32000/60000]
loss: 0.679932 [38400/60000]
loss: 0.687115 [44800/60000]
loss: 0.760144 [51200/60000]
loss: 0.502617 [57600/60000]
Errore valutazione:
 Accuracy: 75.9%, errore medio: 0.683108

Fatto!
```

THANKS!

IR0000032 – ITINERIS, Italian Integrated Environmental Research Infrastructures System
(D.D. n. 130/2022 - CUP B53C22002150006) Funded by EU - Next Generation EU PNRR-Mission 4 “Education and Research” - Component 2: “From research to business” - Investment 3.1: “Fund for the realisation of an integrated system of research and innovation infrastructures”

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani